Package: fmridesign (via r-universe)

June 2, 2026

Type Package

Title Design Matrix Construction for fMRI Analysis

Version 0.6.0

Date 2026-05-12

Description Constructs and inspects design matrices for functional magnetic resonance imaging (fMRI) analyses. Provides a formula interface for specifying event-related designs with flexible hemodynamic response function (HRF) bases, parametric modulators, and trialwise models. Includes facilities for baseline/nuisance regressors, contrast specification, and utilities for naming, validation, and visualization. Intended to support the model-building stage prior to statistical fitting. Implements methods described in Friston et al. (1995) <doi:10.1006/nimg.1995.1007> and Lindquist (2008) <doi:10.1214/09-STS282>.

License MIT + file LICENSE

Encoding UTF-8

LazyData false

Depends R (>= 4.1.0)

Imports fmrihrf (>= 0.3.0), stats, assertthat, rlang, stringr, dplyr, tidyr, purrr, tibble, Matrix, splines, plotly, ggplot2, utils, cli

Suggests testthat, knitr, rmarkdown, covr, pkgdown, withr

VignetteBuilder knitr

Config/Needs/website github::bbuchsbaum/albersdown, albersdown

URL https://github.com/bbuchsbaum/fmridesign, https://bbuchsbaum.github.io/fmridesign/

BugReports https://github.com/bbuchsbaum/fmridesign/issues

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

Collate 'baseline_model.R' 'basis.R' 'condition_basis_list.R'
'contrast.R' 'contrast_pipeline.R' 'covariate.R'
'design_colmap.R' 'design_generics.R' 'design_map.R'
'design_metadata.R' 'utils-internal.R' 'event-classes.R'
'event_model_helpers.R' 'event_model.R' 'event_vector.R'
'extension_registry.R' 'fmrihrf-imports.R'
'fmrihrf-reexports.R' 'globals.R' 'hrf-formula.R'
'namespace-imports.R' 'naming-utils.R' 'residualize.R'
'sampling_frame.R' 'validate.R' 'zzz.R'

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev

Repository https://bbuchsbaum.r-universe.dev

Date/Publication 2026-06-02 18:40:14 UTC

RemoteUrl https://github.com/bbuchsbaum/fmridesign

RemoteRef HEAD

RemoteSha 89a475f2a49a1dfa15a9a0771b981e635ef7041c

Contents

`baseline` *Create a Baseline Specification*

Description

Generates a baselinespec for modeling low-frequency drift in fMRI time series.

Usage

```
baseline(
  degree = 1,
  basis = c("constant", "poly", "bs", "ns"),
  name = NULL,
  intercept = c("runwise", "global", "none")
)
```

Arguments

degree	Number of basis terms per image block (ignored for "constant").
basis	Type of basis ("constant", "poly", "bs", or "ns").
name	Optional name for the term.
intercept	Type of intercept to include ("runwise", "global", or "none").

Value

A baselinespec list instance.

Examples

```
baseline(degree = 3, basis = "bs")
```

---

baseline_model *Construct a Baseline Model*

Description

Builds a baseline model to account for noise and non-event-related variance. This model may include a drift term, a block intercept term, and nuisance regressors.

Usage

```
baseline_model(
  basis = c("constant", "poly", "bs", "ns"),
  degree = 1,
  sframe,
  intercept = c("runwise", "global", "none"),
  nuisance_list = NULL,
  nuisance_check = c("warn", "error", "drop", "none"),
  na_action = c("drop", "zero", "median")
)
```

Arguments

basis	Character; type of basis function ("constant", "poly", "bs", or "ns").
degree	Integer; degree of the spline/polynomial function.
sframe	A sampling_frame object.
intercept	Character; whether to include an intercept ("runwise", "global", or "none"). Ignored when `basis == "constant"` because the drift term already provides the constant baseline.
nuisance_list	Optional list of nuisance matrices or data frames (one per fMRI block).

nuisance_check	Character; how to handle nuisance diagnostics. "warn" warns on construction-time problems, "error" stops, "drop" removes non-finite, zero-variance, and rank-aliased columns with a warning, and "none" skips these checks.
na_action	Character; how to handle NA values in nuisance_list columns before the diagnostics run. "drop" (default) leaves NAs in place so any column containing one is treated as non-finite and removed by nuisance_check. "zero" replaces NA with 0 (matching the fMRIPrep leading-row convention) and "median" replaces NA with the column median; both repair an isolated leading NA (e.g. in DVARS or framewise displacement) so the regressor is retained. NaN and Inf are never repaired and remain subject to the non-finite drop. Repair is applied even when nuisance_check = "none", preventing NAs from leaking into the design matrix.

Value

An object of class "baseline_model".

Examples

```
sframe <- fmrihrf::sampling_frame(blocklens = c(100, 100), TR = 2)
bmod <- baseline_model(basis = "bs", degree = 3, sframe = sframe)
bmod_global <- baseline_model(basis = "bs", degree = 3, sframe = sframe, intercept = "global")
bmod_nointercept <- baseline_model(basis = "bs", degree = 3, sframe = sframe, intercept = "none")
stopifnot(ncol(design_matrix(bmod)) == 8)
```

---

baseline_terms.baseline_model

Extract baseline terms

Description

Extract baseline terms

Usage

```
## S3 method for class 'baseline_model'
baseline_terms(x, ...)

baseline_terms(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

A named list of baseline term objects.

Examples

```
sframe <- fmrihrf::sampling_frame(blocklens = 6, TR = 1)
bmod <- baseline_model(sframe = sframe)
baseline_terms(bmod)
```

`basis_suffix` *Create Basis Function Suffix*

Description

Generates the _b## suffix for HRF basis functions.

Usage

```
basis_suffix(j, nb)
```

Arguments

`j`	Integer vector of basis indices (1-based).
`nb`	Total number of basis functions.

Value

Character vector of suffixes (e.g., `_b01`, `_b02`).

Examples

```
basis_suffix(1:3, 5)
basis_suffix(1:10, 10)
```

`block` *Create a Block Variable*

Description

Returns a block variable that is constant over the span of a scanning run.

Usage

```
block(x)
```

Arguments

`x`	The block variable.

Value

An object of class "blockspec".

Examples

```
block(run)
```

`boxcar_hrf_gen` *Create duration-based boxcar HRF generator*

Description

Creates a generator function for use with the `hrf_fun` parameter in `hrf()`. The generator produces boxcar HRFs where each event's duration determines the boxcar width.

Usage

```
boxcar_hrf_gen(normalize = TRUE, min_duration = 0.1)
```

Arguments

`normalize`	Logical; whether to normalize the boxcar HRF. Default TRUE.
`min_duration`	Numeric; minimum duration to use (prevents zero-width boxcars). Default 0.1.

Value

A function that takes an event data frame and returns a list of HRF objects.

See Also

`weighted_hrf_gen()` for weighted impulse HRFs

Examples

```
trial_data <- data.frame(
  onset = c(0, 10, 25),
  duration = c(2, 5, 3),
  condition = c("A", "B", "A"),
  run = 1
)
sf <- fmrihrf::sampling_frame(blocklens = 50, TR = 2)

emod <- event_model(
  onset ~ hrf(condition, hrf_fun = boxcar_hrf_gen()),
  data = trial_data, block = ~run, sampling_frame = sf, durations = trial_data$duration
)
print(emod)
```

BSpline *B-spline basis*

Description

Generate the B-spline basis matrix for a polynomial spline.

Usage

```
BSpline(x, degree)
```

Arguments

x	a numeric vector at which to evaluate the spline. Missing values are not allowed in x
degree	the degree of the piecewise polynomial

Value

an BSpline list instance

See Also

bs

Examples

```
x_vals <- seq(0, 1, length.out = 6)
bs_obj <- BSpline(x_vals, degree = 3)
dim(bs_obj$y)
bs_obj$name
```

cells.baseline_model *Extract cells from a design object*

Description

Extract cells from a design object

Usage

```
## S3 method for class 'baseline_model'
cells(x, drop.empty = TRUE, ...)

cells(x, drop.empty = TRUE, ...)

## S3 method for class 'event'
cells(x, drop.empty = TRUE, ...)

## S3 method for class 'event_term'
cells(x, drop.empty = TRUE, ...)

## S3 method for class 'covariate_convolved_term'
cells(x, ...)
```

Arguments

x	The object to extract cells from.
drop.empty	Logical indicating whether to drop empty cells (default: TRUE).
...	Additional arguments (e.g., exclude_basis for convolved_term method).

Value

A data.frame/tibble of cells (categorical combinations) relevant to x.

Examples

```
sframe <- fmrihrf::sampling_frame(blocklens = 6, TR = 1)
bmod <- baseline_model(sframe = sframe)
head(cells(bmod))
```

---

check_collinearity *Check design matrix for multicollinearity*

Description

Convenience helper to quickly flag highly correlated regressors.

Usage

```
check_collinearity(X, threshold = 0.9)
```

Arguments

X	A numeric design matrix (or an event_model).
threshold	Absolute correlation above which a pair is flagged. Default 0.9.

Value

A list with elements: ok (logical), pairs (data.frame with offending pairs and their correlations). Invisibly returns the same list.

Examples

```
# Create a simple event model
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emodel <- event_model(onset ~ hrf(cond), data = des, block = ~run,
                      sampling_frame = sframe)

# Check for multicollinearity
res <- check_collinearity(design_matrix(emodel), threshold = 0.95)
if (!res$ok) print(res$pairs)
```

---

check_nuisance *Check nuisance regressors for rank and column problems*

Description

Inspects a block-wise nuisance regressor list before it is added to a baseline model. The check is run per block and compares nuisance columns against the baseline terms that would be constructed from basis, degree, and intercept.

Usage

```
check_nuisance(
  nuisance_list,
  sframe,
  basis = c("constant", "poly", "bs", "ns"),
  degree = 1,
  intercept = c("runwise", "global", "none"),
  tol = sqrt(.Machine$double.eps),
  duplicate_threshold = 1 - sqrt(.Machine$double.eps),
  na_action = c("drop", "zero", "median")
)
```

Arguments

nuisance_list	A list of numeric matrices or data frames, one per block.
sframe	A sampling frame.
basis, degree, intercept	Baseline model settings used to construct the comparison baseline terms.

`tol` Numeric tolerance passed to QR rank checks.

`duplicate_threshold`
Absolute correlation threshold used to flag duplicate or near-duplicate columns.

`na_action` Character; how to handle `NA` values in `nuisance_list` columns before the diagnostics run. `"drop"` (default) leaves `NA`s in place so any column containing one is treated as non-finite. `"zero"` replaces `NA` with `0` (matching the fMRIPrep leading-row convention) and `"median"` replaces `NA` with the column median; both repair an isolated leading `NA` (e.g. in DVARS or framewise displacement) so the regressor is retained. `NaN` and `Inf` are never repaired and remain non-finite.

Value

A `nuisance_check` object with `ok`, `problems`, `by_block`, and normalized `nuisance_list` elements.

`clean_nuisance` *Clean nuisance regressors by dropping rank-useless columns*

Description

Drops nuisance columns that are non-finite, zero-variance, or fail to increase QR rank after the block's baseline terms and earlier nuisance columns. Column order is respected, so when two columns are aliased the earlier column is kept.

Usage

```
clean_nuisance(
  nuisance_list,
  sframe,
  basis = c("constant", "poly", "bs", "ns"),
  degree = 1,
  intercept = c("runwise", "global", "none"),
  tol = sqrt(.Machine$double.eps),
  duplicate_threshold = 1 - sqrt(.Machine$double.eps),
  na_action = c("drop", "zero", "median")
)
```

Arguments

`nuisance_list` A list of numeric matrices or data frames, one per block.

`sframe` A sampling frame.

`basis`, `degree`, `intercept`
Baseline model settings used to construct the comparison baseline terms.

`tol` Numeric tolerance passed to QR rank checks.

duplicate_threshold
: Absolute correlation threshold used to flag duplicate or near-duplicate columns.

na_action
: Character; how to handle NA values in nuisance_list columns before the diagnostics run. "drop" (default) leaves NAs in place so any column containing one is treated as non-finite. "zero" replaces NA with 0 (matching the fMRIPrep leading-row convention) and "median" replaces NA with the column median; both repair an isolated leading NA (e.g. in DVARS or framewise displacement) so the regressor is retained. NaN and Inf are never repaired and remain non-finite.

Value

A list with nuisance_list and report elements. Pass result$nuisance_list to baseline_model().

---

column_contrast *Column Contrast Specification*

Description

Define a contrast by directly targeting design matrix columns using regex patterns. This is useful for contrasts involving continuous variables or specific basis functions.

Usage

```
column_contrast(pattern_A, pattern_B = NULL, name, where = NULL)
```

Arguments

pattern_A
: A character string containing a regex pattern to identify the columns for the positive (+) part of the contrast.

pattern_B
: Optional character string containing a regex pattern for the negative (-) part (for A-B type contrasts). If NULL, creates a contrast testing the average of columns matching pattern_A against baseline (0).

name
: A character string name for the contrast (mandatory).

where
: Currently unused for column_contrast, but kept for API consistency.

Details

This contrast type operates by finding design matrix columns whose names match the provided patterns (pattern_A, pattern_B). It calculates weights such that the average effect of the 'A' columns is compared to the average effect of the 'B' columns (or baseline if pattern_B is NULL). Weights are assigned as +1/nA for 'A' columns and -1/nB for 'B' columns, ensuring the contrast sums to zero if both A and B groups are present.

Use standard R regex syntax for the patterns. Remember to escape special characters (e.g., \\[, \\., *).

Value

A column_contrast_spec object containing the specification.

Examples

```
# Test the main effect of a continuous modulator 'RT'
# Assumes RT is a column name, e.g., from columns(Scale(RT))
cc1 <- column_contrast(pattern_A = "^z_RT$", name = "Main_RT")

# Compare Condition.A vs Condition.B for the 'RT' modulator effect
# Assumes condition names like "Condition.A_z_RT", "Condition.B_z_RT"
cc2 <- column_contrast(pattern_A = "^Condition\\.A_z_RT$",
                       pattern_B = "^Condition\\.B_z_RT$",
                       name = "CondA_vs_CondB_for_RT")

# Test a specific basis function (e.g., basis spline #3)
# Assumes column names like "TermName_Condition.Tag_b03"
cc3 <- column_contrast(pattern_A = "_b03$", name = "Basis_3_Effect")
```

---

columns.Scale *Extract columns*

Description

Extract columns

Usage

```
## S3 method for class 'Scale'
columns(x, ...)

## S3 method for class 'ScaleWithin'
columns(x, ...)

## S3 method for class 'RobustScale'
columns(x, ...)

columns(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

Character vector of column names produced by the object.

Examples

```
bs_basis <- BSpline(seq(0, 1, length.out = 5), degree = 3)
columns(bs_basis)
```

`condition_basis_list` *Convert an* `event_term` *to a per-condition basis list*

Description

A lightweight wrapper around `convolve()` that post-processes the resulting design matrix into a named list of T x d matrices - one per experimental condition ("base condition tag"). This keeps **all** of the heavy lifting inside **fmrireg** while exposing a minimal, pipe-friendly API that can be used anywhere a condition -> basis split is required (e.g. for CFALS).

Usage

```
condition_basis_list(
  x,
  hrf,
  sampling_frame,
  ...,
  output = c("condition_list", "matrix")
)
```

Arguments

`x`	An `event_term` object.
`hrf`	An `HRF` object to apply.
`sampling_frame`	A `sampling_frame` object defining the temporal grid.
`...`	Further arguments passed on to `convolve()` (e.g. `drop.empty = FALSE`).
`output`	Either "matrix" (default) for the ordinary design matrix or "condition_list" for the split-by-condition list.

Value

A numeric *matrix* or a named *list* of matrices, depending on `output`.

Examples

```
term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
sf <- fmrihrf::sampling_frame(blocklens = 30, TR = 1)
condition_basis_list(term, fmrihrf::HRF_SPMG1, sf)
```

condition_map *Map Display and Canonical Condition Names*

Description

Map Display and Canonical Condition Names

Usage

```
condition_map(x, drop.empty = TRUE, expand_basis = FALSE, ...)
```

Arguments

x	The object to inspect.
drop.empty	Logical whether to drop empty conditions (default: TRUE).
expand_basis	Logical whether to expand basis functions (default: FALSE).
...	Additional arguments.

Value

A tibble mapping display names to canonical names.

Examples

```
term <- event_term(
  list(
    category = factor(c("face", "scene", "face")),
    attention = factor(c("attend", "attend", "ignore"))
  ),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
condition_map(term)
```

conditions *Extract conditions from a design object*

Description

Extract conditions from a design object

Usage

```
conditions(
  x,
  drop.empty = TRUE,
  expand_basis = FALSE,
  style = c("canonical", "display"),
  ...
)

## S3 method for class 'event_term'
conditions(
  x,
  drop.empty = TRUE,
  expand_basis = FALSE,
  style = c("canonical", "display"),
  ...
)
```

Arguments

x	The object to extract conditions from.
drop.empty	Logical whether to drop conditions with no events (default: TRUE).
expand_basis	Logical whether to expand basis functions (default: FALSE).
style	Naming style. "canonical" returns fully qualified internal names, while "display" returns shorter user-facing labels.
...	Additional arguments.

Value

A character vector of condition names.

Examples

```
term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
conditions(term)
conditions(term, style = "display")
conditions(term, expand_basis = TRUE)
```

construct.baselinespec *Construct method*

Description

Construct method

Usage

```
## S3 method for class 'baselinespec'
construct(x, model_spec, ...)

## S3 method for class 'covariatespec'
construct(x, model_spec, sampling_frame = NULL, ...)

construct(x, ...)
```

Arguments

x	The object.
model_spec	A model specification object (used by some methods). For baselinespec: typically a sampling_frame or list containing one. For hrfspec/covariatespec: contains data and other model information.
...	Additional arguments.
sampling_frame	A sampling_frame object (used by covariatespec method).

Value

A constructed object; return type depends on method.

Examples

```
sframe <- fmrihrf::sampling_frame(blocklens = 5, TR = 1)
drift_spec <- baseline(degree = 2, basis = "poly")
construct(drift_spec, sframe)
```

contrast *Contrast Specification*

Description

Define a linear contrast using a formula expression.

Usage

```
contrast(form, name, where = NULL)
```

Arguments

form	A formula describing the contrast.
name	A character label for the contrast.
where	An expression defining the subset over which the contrast is applied (default: NULL).

Value

A list containing the contrast specification.

Examples

```
# A minus B contrast using display labels
contrast(~ A - B, name="A_B")

# With subsetting
contrast(~ A - B, name="A_B_block1", where = ~ block == 1)
```

contrast_from_mask *Package a base mask into a* contrast *object*

Description

Drives the unified post-processing pipeline: expands the base mask across basis functions (when applicable), applies basis filtering, attaches the standard contrast list class, and returns an object compatible with downstream contrast_weights.event_model().

Usage

```
contrast_from_mask(mask, spec, term, classes = character())
```

Arguments

mask	Output of contrast_mask() (or compatible list).
spec	The originating contrast spec (stored on the result).
term	The event_term the contrast is being computed against.
classes	Character vector of extra S3 classes prepended to the default c("cell_contrast", "contrast", "list").

Details

Custom contrast types implement contrast_mask() and call this driver.

Value

A contrast object.

---

contrast_mask *Build a base contrast mask*

Description

contrast_mask() is the extension point for new contrast types: it returns a *base* contrast matrix with one row per base condition (no basis expansion) and one column per contrast column. The unified driver contrast_from_mask() handles basis expansion, basis filtering and packaging into a contrast object compatible with contrast_weights.event_model().

Usage

```
contrast_mask(x, term, ...)
```

Arguments

x	A contrast_spec object.
term	An event_term against which the mask is computed.
...	Additional arguments passed to methods.

Details

To register a custom contrast type:

1. Define a constructor that produces a list with class c("my_spec", "contrast_spec", "list").
2. Implement contrast_mask.my_spec() returning the base weights.
3. Implement contrast_weights.my_spec() as a one-liner:

```
contrast_weights.my_spec <- function(x, term, ...) {
  contrast_from_mask(contrast_mask(x, term, ...), x, term)
}
```

The built-in spec classes (pair_contrast_spec, oneway_contrast_spec, poly_contrast_spec, ...) implement contrast_weights() directly, so they do not require contrast_mask() methods.

Value

A list with elements:

- `weights` : numeric matrix, `n_base_conditions x n_contrast_cols`, row names = base condition names.
- `condnames` : character vector of base condition names.

`contrast_set` *Create a Set of Contrasts*

Description

Construct a list of contrast_spec objects.

Usage

```
contrast_set(...)
```

Arguments

...	A variable-length list of contrast_spec objects.

Value

A list of contrast_spec objects with class "contrast_set".

Examples

```
c1 <- contrast(~ A - B, name="A_B")
c2 <- contrast(~ B - C, name="B_C")
contrast_set(c1,c2)
```

`contrast_weights.unit_contrast_spec`
Unit Contrast Weights

Description

Compute the contrast weights for a unit_contrast_spec object.

Compute the contrast weights for an oneway_contrast_spec object.

Compute the contrast weights for an interaction_contrast_spec object.

Compute the contrast weights for a poly_contrast_spec object.

Compute the contrast weights for a pair_contrast_spec object.

Compute contrast weights for a `column_contrast_spec` object by targeting design matrix columns based on regex patterns.

Compute the contrast weights for a contrast_formula_spec object.

Compute the contrast weights for a contrast_diff_spec object.

Compute the contrast weights for each contrast specification within a contrast_set object.

Usage

```
## S3 method for class 'unit_contrast_spec'
contrast_weights(x, term, ...)

## S3 method for class 'oneway_contrast_spec'
contrast_weights(x, term, ...)

## S3 method for class 'interaction_contrast_spec'
contrast_weights(x, term, ...)

## S3 method for class 'poly_contrast_spec'
contrast_weights(x, term, ...)

## S3 method for class 'pair_contrast_spec'
contrast_weights(x, term, ...)

## S3 method for class 'column_contrast_spec'
contrast_weights(x, term, ...)

## S3 method for class 'contrast_formula_spec'
contrast_weights(x, term, ...)

## S3 method for class 'contrast_diff_spec'
contrast_weights(x, term, ...)

## S3 method for class 'contrast_set'
contrast_weights(x, term, ...)

contrast_weights(x, ...)

## S3 method for class 'convolved_term'
contrast_weights(x, ...)
```

```
## S3 method for class 'event_model'
contrast_weights(x, ...)
```

Arguments

x	The object.
term	A term object against which weights should be computed.
...	Additional arguments.

Details

If the weight matrices returned by a contrast specification contain row names, these are matched to the column names of the corresponding term in the design matrix. This allows contrasts to target only a subset of term levels.

Value

A list containing the term, name, weights, condition names, and contrast specification.

A list containing the term, name, weights, condition names, and contrast specification.

A list containing the term, name, weights, condition names, and contrast specification.

A list containing the term, name, weights, condition names, and contrast specification.

A list containing the term, name, weights, condition names, and contrast specification.

A list containing the contrast details:

term	The original event_term object.
name	The name of the contrast.
weights	A numeric matrix where rows correspond to the full design matrix columns (from .condnames(term, expanded = TRUE)) and columns represent the contrast(s). Usually one column.
condnames	Character vector of all potential *expanded* condition names from term.
contrast_spec	The original column_contrast_spec object.

A list containing the term, name, weights, condition names, and contrast specification.

A list containing the term, name, weights, condition names, and contrast specification.

A named list where each element is the result of calling contrast_weights on the corresponding contrast_spec in the set. The list names are the names of the individual contrasts.

A named list of contrast weight objects or matrices.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
```

```
cset <- contrast_set(
  diff = column_contrast(pattern_A = "cond.A", pattern_B = "cond.B", name = "diff")
)
emod <- event_model(onset ~ hrf(cond, contrasts = cset),
                    data = des, block = ~run, sampling_frame = sframe)
contrast_weights(emod)
```

`contrasts` *Extract contrasts*

Description

Extract contrasts

Usage

```
contrasts(x, ...)
```

Arguments

`x`	The object.
`...`	Additional arguments.

Value

A named list of contrast specifications.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
cset <- contrast_set(
  diff = column_contrast(pattern_A = "cond.A", pattern_B = "cond.B", name = "diff")
)
emod <- event_model(onset ~ hrf(cond, contrasts = cset),
                    data = des, block = ~run, sampling_frame = sframe)
contrasts(emod)
```

contrasts.event_model *Retrieve contrast definitions for an event model*

Description

This function collects the contrast specifications defined for each term within the model and returns them as a named list.

Usage

```
## S3 method for class 'event_model'
contrasts(x, ...)
```

Arguments

x	An event_model object.
...	Unused.

Value

A named list of contrast specifications.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
cset <- contrast_set(
  diff = column_contrast(pattern_A = "cond.A", pattern_B = "cond.B", name = "diff")
)
emod <- event_model(onset ~ hrf(cond, contrasts = cset),
                    data = des, block = ~run, sampling_frame = sframe)
contrasts(emod)
```

contrasts.event_term *Retrieve contrast definitions for an event term*

Description

This accessor returns the list of contrast specifications attached to the term's originating hrfspec, if any.

Usage

```
## S3 method for class 'event_term'
contrasts(x, ...)
```

Arguments

x	An event_term object.
...	Unused.

Value

A list of contrast specifications or NULL when none are defined.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
cset <- contrast_set(
  diff = column_contrast(pattern_A = "cond.A", pattern_B = "cond.B", name = "diff")
)
emod <- event_model(onset ~ hrf(cond, contrasts = cset),
                    data = des, block = ~run, sampling_frame = sframe)
contrasts(terms(emod)[[1]])
```

---

contrasts.hrfspec *Retrieve contrast specifications from an hrfspec*

Description

Retrieve contrast specifications from an hrfspec

Usage

```
## S3 method for class 'hrfspec'
contrasts(x, ...)
```

Arguments

x	An hrfspec object.
...	Unused.

Value

The list of contrast specifications attached to the hrfspec, or NULL.

Examples

```
condition <- factor(c("A", "B"))
cset <- contrast_set(
 diff = column_contrast(pattern_A = "condition.A", pattern_B = "condition.B", name = "diff")
)
spec <- hrf(condition, contrasts = cset)
contrasts(spec)
```

`convolve` *Convolve events with a hemodynamic response function*

Description

Convolve events with a hemodynamic response function

Usage

```
convolve(
  x,
  hrf,
  sampling_frame,
  drop.empty = TRUE,
  summate = TRUE,
  precision = 0.1,
  normalize = FALSE,
  ...
)

## S3 method for class 'event_term'
convolve(
  x,
  hrf,
  sampling_frame,
  drop.empty = TRUE,
  summate = TRUE,
  precision = 0.3,
  normalize = FALSE,
  ...
)
```

Arguments

`x`	The events to convolve.
`hrf`	The hemodynamic response function.
`sampling_frame`	The sampling frame.
`drop.empty`	Logical indicating whether to drop columns with all zeros.

summate	Logical indicating whether to sum convolved signals.
precision	Numeric specifying the temporal precision for convolution.
normalize	Logical; if TRUE, each convolved regressor column is peak-normalized. Default FALSE.
...	Additional arguments.

Value

A matrix-like (often tibble) of convolved regressors.

Examples

```
term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 5, 10),
  blockids = c(1, 1, 1)
)
sf <- fmrihrf::sampling_frame(blocklens = 20, TR = 1)
conv <- convolve(term, fmrihrf::HRF_SPMG1, sf)
names(conv)
```

---

convolve_design *Convolve HRF with Design Matrix.*

Description

Convolves a HRF with a design matrix (one column per condition) to produce a list of regressors.

Usage

```
convolve_design(hrf, dmat, globons, durations, summate = TRUE, hrf_list = NULL)
```

Arguments

hrf	A function representing the HRF (used when hrf_list is NULL).
dmat	Design matrix (with named columns).
globons	Numeric vector of global onsets.
durations	Numeric vector of event durations.
summate	Logical; if TRUE, summate the convolved HRF (default: TRUE).
hrf_list	Optional list of HRF objects, one per event (row in dmat). When provided, allows per-onset HRF specification. The list is subsetted to match non-zero amplitude events for each condition.

Value

A list of regressors (one for each column).

Examples

```
hrf <- fmrihrf::HRF_SPMG1
dmat <- data.frame(A = c(1, 0, 1), B = c(0, 1, 0))
globons <- c(0, 10, 20)
durations <- rep(0, 3)
regs <- convolve_design(hrf, dmat, globons, durations)
length(regs)
```

---

correlation_map *Compute correlation map*

Description

Compute correlation map

Usage

```
correlation_map(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

A ggplot2 object visualizing regressor correlations.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)
correlation_map(emod)
```

correlation_map.baseline_model
correlation_map.baseline_model

Description

Generates a correlation heatmap of the columns in a `baseline_model`'s design matrix.

Usage

```
## S3 method for class 'baseline_model'
correlation_map(
  x,
  method = c("pearson", "spearman"),
  half_matrix = FALSE,
  absolute_limits = TRUE,
  ...
)
```

Arguments

`x`	A `baseline_model`.
`method`	Correlation method (e.g., "pearson", "spearman").
`half_matrix`	Logical; if TRUE, display only the lower triangle of the matrix.
`absolute_limits`	Logical; if TRUE, set color scale limits from -1 to 1.
`...`	Additional arguments passed to internal plotting functions.

Value

A ggplot2 plot object.

Examples

```
sframe <- fmrihrf::sampling_frame(blocklens = 5, TR = 1)
bmod <- baseline_model(sframe = sframe)
if (requireNamespace("ggplot2", quietly = TRUE)) correlation_map(bmod)
```

correlation_map.event_model *Visualize Regressor Correlations*

Description

Creates a heatmap visualization of the correlation matrix between regressors in an event_model object.

Usage

```
## S3 method for class 'event_model'
correlation_map(x, rotate_x_text = TRUE, ...)
```

Arguments

x	An event_model object.
rotate_x_text	Logical. Whether to rotate x-axis labels. Default is TRUE.
...	Additional arguments passed to geom_tile.

Value

A ggplot2 object showing the correlation matrix heatmap.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)
correlation_map(emod)
```

covariate *Construct a Covariate Term*

Description

Creates a covariate term that is added directly to the fMRI model without being convolved with a hemodynamic response function (HRF). This is useful for including nuisance variables, continuous covariates, or any other regressors that should not undergo HRF convolution.

Usage

```
covariate(..., data, id = NULL, prefix = NULL, subset = NULL)
```

Arguments

...	A variable argument set of covariate names.
data	A data.frame containing the variables.
id	An optional identifier for the covariate term.
prefix	An optional prefix to add to the covariate names.
subset	Optional expression used to subset the covariate data.

Details

In fMRI analysis, some predictors should not be convolved with the HRF because they represent:

- Continuous physiological measurements (e.g., heart rate, respiration)
- Motion parameters from head movement correction
- Scanner drift or other technical artifacts
- Behavioral measures that directly correlate with BOLD signal
- Global signal or other nuisance variables

The covariate term can be combined with standard HRF-convolved event terms in the same model. For example:

```
model <- event_model(onset ~ hrf(stimulus) + covariate(motion_x, motion_y, data = cov_data),
                     data = events, block = ~ 1, sampling_frame = sframe)
```

Value

A list containing information about the covariate term with class 'covariatespec' that can be used within an event_model.

See Also

- event_model() for creating complete fMRI models
- hrf() for creating HRF-convolved event terms

Examples

```
# Add motion parameters as covariates
motion_data <- data.frame(
  x = rnorm(100),  # x translation
  y = rnorm(100)   # y translation
)
cv <- covariate(x, y, data = motion_data, prefix = "motion")

# Combine with event model
sframe <- sampling_frame(blocklens = c(100), TR = 2)
# 50 events, strictly increasing onsets per block
event_data <- data.frame(
  stimulus = factor(rep(c("A", "B"), 25)),
  onset = seq(0, by = 4, length.out = 50)
```

```
)

# Full model with both HRF-convolved events and non-convolved covariates
model <- event_model(
  onset ~ hrf(stimulus) + covariate(x, y, data = motion_data, id = "motion"),
  data = event_data,
  block = ~ 1,
  sampling_frame = sframe
)
```

`design_colmap` *Column Metadata Map*

Description

Returns a tibble with one row per column of the design matrix and structured metadata (term, condition, basis, role, run, etc.).

Usage

```
design_colmap(x, ...)
```

Arguments

`x`	An object containing or producing a design matrix (e.g., event_model, baseline_model).
`...`	Additional arguments passed to methods.

Value

A tibble with per-column metadata.

Examples

```
# Create a simple event model
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)

# Extract column metadata
colmap <- design_colmap(emod)
head(colmap)

# Query specific columns
colmap[colmap$condition == "A", ]
```

`design_colmap.event_model` *Column metadata for a design matrix*

Description

Returns a tibble with one row per column of the design matrix and structured metadata derived from existing package attributes and naming conventions. This avoids ad-hoc string parsing in user code and provides a consistent, queryable view of the design.

Usage

```
## S3 method for class 'event_model'
design_colmap(x, ...)
```

Arguments

`x`	An object containing or producing a design matrix (e.g., `event_model`, `baseline_model`).
`...`	Unused; reserved for future extensions.

Value

A tibble with columns:

- `col` (integer): 1-based column index
- `name` (character): column name
- `term_tag` (character): term identifier (event tag or baseline term name)
- `term_index` (integer): position within `terms(x)`
- `condition` (character): condition/event label without basis suffix (if applicable)
- `run` (integer): block/run id when the column is block-specific; `NA` if pooled
- `role` (character): e.g., "task", "drift", "intercept", "nuisance"
- `model_source` (character): "event" or "baseline"
- `basis_name` (character): HRF or baseline basis identifier when available
- `basis_ix` (integer): within-condition basis index (HRF component); `NA` if not applicable
- `basis_total` (integer): total number of basis components for the column's term; `NA` if not applicable
- `basis_label` (character): human-readable label for the basis component when known
- `pretty_name` (character): concise, human-friendly label for display (e.g., "RT" or "RT_lag_02")
- `is_block_diagonal` (logical): TRUE when the regressor is per-run/block
- `modulation_type` (character): "amplitude", "parametric", or "covariate"
- `modulation_id` (character): modulator identifier when applicable (e.g., "RT", "RT_by_group")

Examples

```
# Create event model
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)

# Get column metadata
colmap <- design_colmap(emod)
head(colmap)

# Query columns by condition
colmap[colmap$condition == "A", ]
```

`design_map` *Compute design map*

Description

Compute design map

Usage

```
design_map(x, ...)
```

Arguments

`x`	The object.
`...`	Additional arguments.

Value

A ggplot2 object visualizing the design matrix.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)
design_map(emod)
```

`design_map.baseline_model` *Heatmap visualization of the baseline_model design matrix*

Description

Produces a heatmap of all columns in the design matrix for a `baseline_model` object, with rows corresponding to scans and columns corresponding to regressors. By default, it draws horizontal lines separating runs (blocks), and rotates the column labels diagonally.

Usage

```
## S3 method for class 'baseline_model'
design_map(
  x,
  block_separators = TRUE,
  rotate_x_text = TRUE,
  fill_midpoint = NULL,
  fill_limits = NULL,
  ...
)
```

Arguments

`x`	A `baseline_model` object.
`block_separators`	Logical; if `TRUE`, draw white horizontal lines between blocks.
`rotate_x_text`	Logical; if `TRUE`, rotate x-axis labels by 45 degrees.
`fill_midpoint`	Numeric or `NULL`; if not `NULL`, used as the `midpoint` in `ggplot2::scale_fill_gradient2()` to center the color scale (for example at 0).
`fill_limits`	Numeric vector of length 2 or `NULL`; passed to the fill scale `limits` argument. Can clip or expand the color range.
`...`	Additional arguments forwarded to `ggplot2::geom_tile()`.

Value

A ggplot2 plot object.

Examples

```
sframe <- fmrihrf::sampling_frame(blocklens = 5, TR = 1)
bmod <- baseline_model(sframe = sframe)
if (requireNamespace("ggplot2", quietly = TRUE)) design_map(bmod)
```

`design_map.event_model` *Visualize Event Model Design Matrix*

Description

Creates a heatmap visualization of the design matrix for an event_model object.

Usage

```
## S3 method for class 'event_model'
design_map(
  x,
  block_separators = TRUE,
  rotate_x_text = TRUE,
  fill_midpoint = NULL,
  fill_limits = NULL,
  ...
)
```

Arguments

`x`	An `event_model` object.
`block_separators`	Logical. Whether to draw separators between blocks/runs. Default is TRUE.
`rotate_x_text`	Logical. Whether to rotate x-axis labels. Default is TRUE.
`fill_midpoint`	Numeric. Midpoint for color scale. If NULL, uses gradient scale.
`fill_limits`	Numeric vector of length 2. Limits for fill scale.
`...`	Additional arguments passed to geom_tile.

Value

A ggplot2 object showing the design matrix heatmap.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)
design_map(emod)
```

`design_matrix.baseline_model` *Extract or construct a design matrix*

Description

Extract or construct a design matrix

Usage

```
## S3 method for class 'baseline_model'
design_matrix(x, blockid = NULL, allrows = FALSE, ...)

## S3 method for class 'baseline_term'
design_matrix(x, blockid = NULL, allrows = FALSE, ...)

design_matrix(x, ...)

## S3 method for class 'event_model'
design_matrix(x, blockid = NULL, ...)

## S3 method for class 'event_term'
design_matrix(x, drop.empty = TRUE, ...)
```

Arguments

`x`	The object to extract design matrix from.
`blockid`	Block ID(s) to extract (for baseline_term method).
`allrows`	Whether to return all rows (for baseline_term method).
`...`	Additional arguments.
`drop.empty`	Whether to drop empty columns (for event_term method).

Value

A matrix-like object (often tibble) with rows = scans, cols = regressors.

Examples

```
sframe <- fmrihrf::sampling_frame(blocklens = 6, TR = 1)
bmod <- baseline_model(sframe = sframe)
head(design_matrix(bmod))
```

design_meta *Accessor: column metadata for a design matrix*

Description

Returns the `col_metadata` tibble attached to a design matrix produced by `event_model()`. If the attribute is missing (e.g., on a manually constructed matrix) returns `NULL`.

Usage

```
design_meta(x)
```

Arguments

`x`	A design matrix or any object with a `design_matrix()` method.

Value

A tibble, or `NULL`.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des,
                    block = ~run, sampling_frame = sframe)
design_meta(emod)
```

duration_hrf_gen *Create duration-aware normalized HRF generator*

Description

Generator for use with the `hrf_fun` parameter in `hrf()`. Creates per-onset HRFs by convolving a base HRF with a boxcar of each event's duration, then normalizing to peak = 1. This preserves the temporal shape of the duration-modulated response while equalizing amplitude across events.

Usage

```
duration_hrf_gen(base = fmrihrf::HRF_SPMG1, min_duration = 0)
```

Arguments

base	Base HRF object. Default fmrihrf::HRF_SPMG1.
min_duration	Minimum duration (prevents zero-width events). Default 0.

Value

A function for use with hrf_fun in hrf().

See Also

boxcar_hrf_gen() for duration-based boxcar HRFs

Examples

```
# Events with variable durations
trial_data <- data.frame(
  onset = c(0, 10, 25),
  duration = c(2, 5, 3),
  condition = c("A", "B", "A"),
  run = 1
)
sf <- fmrihrf::sampling_frame(blocklens = 50, TR = 2)

emod <- event_model(
  onset ~ hrf(condition, hrf_fun = duration_hrf_gen()),
  data = trial_data, block = ~run, sampling_frame = sf,
  durations = trial_data$duration
)
print(emod)
```

---

elements *Extract elements from an object*

Description

Extract elements from an object

Usage

```
elements(x, ...)

## S3 method for class 'event'
elements(x, what = c("values", "labels"), transformed = TRUE, ...)

## S3 method for class 'event_term'
elements(x, what = c("values", "labels"), ...)
```

Arguments

x	The object to extract elements from.
...	Additional arguments.
what	Character string specifying what to extract: "values" for numeric/actual values, or "labels" for descriptive labels/names.
transformed	Logical indicating whether to return transformed values. Default is TRUE.

Value

Requested elements; structure depends on method (e.g., numeric values or labels).

Examples

```
# Create an event term with mixed categorical and continuous events
term <- event_term(
  list(
    condition = factor(c("A", "B", "A", "B")),
    intensity = c(1.2, 0.8, 1.5, 0.9)
  ),
  onsets = c(0, 10, 20, 30),
  blockids = c(1, 1, 1, 1)
)

# Extract values (actual numeric/factor codes)
elements(term, what = "values")

# Extract labels (descriptive names/levels)
elements(term, what = "labels")
```

---

event_basis *Create an event set from a ParametricBasis object.*

Description

This is a user-facing wrapper around the internal `event()` constructor, specifically for creating event sequences modulated by a basis set.

Usage

```
event_basis(
  basis,
  name = NULL,
  onsets,
  blockids = 1,
  durations = 0,
  subset = NULL
)
```

Arguments

basis	A ParametricBasis object (e.g., from BSpline, PolynomialBasis).
name	Optional name for the event variable. If NULL, uses basis$name.
onsets	Numeric vector of event onsets (seconds).
blockids	Numeric vector of block IDs.
durations	Numeric vector of event durations (seconds), or a scalar.
subset	Optional logical vector indicating which events to keep. If provided, the vector must match onsets in length and contain no NA values.

Value

An S3 object of class event and event_seq.

Examples

```
basis <- BSpline(1:21, 3)
onsets <- seq(0, 20, length.out = 21)
blockids <- rep(1, length(onsets))
ebasis <- event_basis(basis, onsets=onsets, blockids=blockids)
print(ebasis)
levels(ebasis)
```

---

event_conditions *Retrieve per-event condition assignments*

Description

Retrieve per-event condition assignments

Usage

```
event_conditions(x, drop.empty = FALSE, ...)
```

Arguments

x	The object of interest.
drop.empty	Logical; drop unused levels when TRUE.
...	Additional arguments passed to methods.

Value

Typically a factor aligned with the events of x.

Examples

```
term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
event_conditions(term)
```

`event_factor` *Create a categorical event sequence from a factor.*

Description

This is a user-facing wrapper around the internal `event()` constructor, specifically for creating categorical event sequences from factors or characters.

Usage

```
event_factor(fac, name, onsets, blockids = 1, durations = 0, subset = NULL)
```

Arguments

`fac`	A factor or something coercible to a factor.
`name`	Name of the event variable.
`onsets`	Numeric vector of event onsets (seconds).
`blockids`	Numeric vector of block IDs.
`durations`	Numeric vector of event durations (seconds), or a scalar.
`subset`	Optional logical vector indicating which events to keep. If provided, the vector must match `onsets` in length and contain no `NA` values. Column names are sanitized using `.sanitizeName()` if provided. If column names are missing or not unique, deterministic feature suffixes (`f01`, `f02`, ...) are generated instead. The resulting names are returned by `levels()` for the event object.

Value

An S3 object of class `event` and `event_seq`.

See Also

`event_model`, `event_variable`, `event_matrix`, `event_basis`

Examples

```
ef_onsets <- seq(1, 100, length.out = 6)
efac <- event_factor(factor(c("a", "b", "c", "a", "b", "c")), "abc",
        onsets = ef_onsets, blockids = rep(1, length(ef_onsets)))
print(efac)
levels(efac)
```

`event_matrix` *Create a continuous event set from a matrix.*

Description

This is a user-facing wrapper around the internal `event()` constructor, specifically for creating continuous event sequences from numeric matrices.

Usage

```
event_matrix(mat, name, onsets, blockids = 1, durations = 0, subset = NULL)
```

Arguments

`mat`	A numeric matrix of continuous event values (one row per event).
`name`	Name of the event variable.
`onsets`	Numeric vector of event onsets (seconds).
`blockids`	Numeric vector of block IDs.
`durations`	Numeric vector of event durations (seconds), or a scalar.
`subset`	Optional logical vector indicating which events to keep. If provided, the vector must match `onsets` in length and contain no `NA` values. If `mat` has column names and more than one column, those names are sanitized using `.sanitizeName()` before being stored. The sanitized column names are returned by `levels()` for the resulting event object.

Value

An S3 object of class `event` and `event_seq`.

Examples

```
mat <- matrix(rnorm(20), 10, 2, dimnames=list(NULL, c("Val1", "Val2")))
onsets <- seq(1, 100, length.out = 10)
durations <- rep(1, 10)
blockids <- rep(1, 10)
eset <- event_matrix(mat, "eset", onsets, blockids, durations)
print(eset)
columns(eset) # Alias for levels
```

`event_model` *Generic functions for fmridesign package*

Description

This file contains the generic functions used throughout the fmridesign package. These generics define the interface for working with event models, baseline models, and related design components. Construct an event model

This is the main constructor for `event_model` objects. It unifies the previous formula and list-based interfaces and uses a more efficient internal pipeline.

Usage

```
event_model(
  formula_or_list,
  data,
  block,
  sampling_frame,
  durations = 0,
  drop_empty = TRUE,
  precision = 0.3,
  parallel = FALSE,
  progress = FALSE,
  strict = FALSE,
  ...
)

event_model(
  formula_or_list,
  data,
  block,
  sampling_frame,
  durations = 0,
  drop_empty = TRUE,
  precision = 0.3,
  parallel = FALSE,
  progress = FALSE,
  strict = FALSE,
  ...
)
```

Arguments

`formula_or_list`
: Either a formula (e.g., `onset ~ hrf(cond) + hrf(mod)`) or a list of pre-defined `hrfspec` objects.

data	A data.frame containing event variables referenced in the formula or needed by the hrfspec objects.
block	A formula (e.g., ~ run) or vector specifying the block/run for each event.
sampling_frame	An object of class sampling_frame defining the scan timing (TR, block lengths).
durations	Numeric vector or scalar specifying event durations (seconds). Default is 0.
drop_empty	Logical indicating whether to drop empty events during term construction. Default is TRUE.
precision	Numeric precision for HRF sampling/convolution. Default is 0.3.
parallel	Logical compatibility shim reserved for future use. The current implementation always evaluates terms sequentially; when set to TRUE, a warning is emitted and sequential evaluation is used. Default is FALSE.
progress	Logical indicating whether to show a progress bar during term realisation. Default is FALSE.
strict	Logical controlling the onset bounds check. When FALSE (default) event onsets that fall outside the sampling frame (negative, at or after a run's end, or events extending past a run's end) trigger a warning(). When TRUE the same conditions raise an error. This is a backstop for gross onset/clock mistakes; a uniform in-bounds shift cannot be detected here.
...	Additional arguments (currently unused).

Details

This function creates an event-based fMRI regression model, represented as a data structure.

Column Naming:

The columns in the resulting design matrix follow the naming convention: term_tag + _ + condition_tag + _b## basis suffix

Where:

- term_tag: The unique tag assigned to the hrf() term (see below).
- condition_tag: Represents the specific factor level or continuous regressor within the term (e.g., condition.A, poly_RT_01, condition.A_task.go).
- _b##: Optional suffix added for HRFs with multiple basis functions (e.g., _b01, _b02).

Term Naming and Clash Resolution:

Each term in the model (typically defined by an hrf() call in a formula) gets a unique term_tag. This tag is used as the prefix for all columns generated by that term.

- **Default Naming:** If no explicit id (or name) is given in hrf(), the tag is derived from the variable names (e.g., hrf(condition) -> condition, hrf(RT, acc) -> RT_acc).
- **Explicit Naming:** Use id= within hrf() for an explicit tag (e.g., hrf(condition, id="CondMain")).
- **Sanitization:** Dots (.) in tags are converted to underscores (_).
- **Clash Resolution:** If multiple terms generate the same tag, # and a number are appended to ensure uniqueness (e.g., condition, condition#1).

This consistent naming scheme replaces the previous compact and qualified styles.

Value

An event_model object describing the task design.

An object of class c("event_model", "list") containing the terms, design matrix, sampling frame, and other metadata.

Examples

```
des <- data.frame(onset = seq(0, 90, by=10),
                  run = rep(1:2, each=5),
                  cond = factor(rep(c("A","B"), 5)),
                  mod = rnorm(10))
sframe <- fmrihrf::sampling_frame(blocklens=c(50, 60), TR=2)

ev_model_form <- event_model(onset ~ hrf(cond) + hrf(mod, basis="spmg3"),
                             data = des, block = ~run, sampling_frame = sframe)
print(ev_model_form)
head(design_matrix(ev_model_form))

spec1 <- hrf(cond)
spec2 <- hrf(mod, basis = "spmg3")
ev_model_list <- event_model(list(spec1, spec2), data = des,
                             block = des$run, sampling_frame = sframe)
print(ev_model_list)
```

---

event_table *Extract event table*

Description

Extract event table

Usage

```
event_table(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

A data.frame/tibble of event rows.

Examples

```
term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
event_table(term)
```

`event_term` *Create an event model term from a named list of variables.*

Description

Generates an `event_term` object which represents the combination of one or more event sequences (e.g., a factor crossed with a numeric modulator). It takes a list of variables (factors, numeric vectors, matrices, basis objects) along with shared onsets, block IDs, and durations. It uses the `EV` factory internally to create standardized `event` objects for each variable.

Usage

```
event_term(evlist, onsets, blockids, durations = 0, subset = NULL)
```

Arguments

`evlist`	A named list of variables (factors, numeric, matrices, ParametricBasis objects). The names are used as variable identifiers within the term.
`onsets`	Numeric vector of onset times (in seconds).
`blockids`	Numeric vector of block numbers (non-decreasing integers).
`durations`	Numeric vector of event durations (seconds, default is 0). Can be scalar (recycled) or vector matching length of `onsets`.
`subset`	Optional logical vector indicating which events to retain (applied before processing).

Value

A list object with class `c("event_term", "event_seq")`. Contains:

`varname`	Concatenated variable names from `evlist`.
`events`	A named list of the processed `event` objects.
`subset`	The `subset` vector used.
`event_table`	A tibble representing the combinations of descriptive levels/names for each event in the term, constructed using `elements(..., values=FALSE)`.
`onsets`	Numeric vector of onsets (after processing/subsetting).
`blockids`	Numeric vector of block IDs (after processing/subsetting).
`durations`	Numeric vector of durations (after processing/subsetting).

Examples

```
x1 <- factor(rep(letters[1:3], 10))
x2 <- factor(rep(1:3, each = 10))
onsets <- seq(1, 100, length.out = 30)
blockids <- rep(1:3, each = 10)

eterm <- event_term(list(Condition = x1, Group = x2),
                    onsets = onsets,
                    blockids = blockids)
print(eterm)
head(event_table(eterm))
levels(eterm)
head(design_matrix(eterm))

term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
head(design_matrix(term))
```

---

event_terms *Extract event terms*

Description

Extract event terms

Usage

```
event_terms(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

A named list of event term objects.

Examples

```
# Create a simple experimental design
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
```

```
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)

# Extract event terms (named list of event term objects)
terms_list <- event_terms(emod)
names(terms_list)
```

`event_variable` *Create a continuous event sequence from a numeric vector.*

Description

This is a user-facing wrapper around the internal `event()` constructor, specifically for creating continuous event sequences from numeric vectors.

Usage

```
event_variable(vec, name, onsets, blockids = 1, durations = 0, subset = NULL)
```

Arguments

`vec`	Numeric vector representing continuous event values.
`name`	Name of the event variable.
`onsets`	Numeric vector of event onsets (seconds).
`blockids`	Numeric vector of block IDs.
`durations`	Numeric vector of event durations (seconds), or a scalar.
`subset`	Optional logical vector indicating which events to keep. If provided, the vector must match `onsets` in length and contain no `NA` values.

Value

An S3 object of class `event` and `event_seq`.

See Also

`event_factor`

Examples

```
ev_onsets <- seq(1, 100, length.out = 6)
evar <- event_variable(c(1, 2, 3, 4, 5, 6), "example_var",
                       onsets = ev_onsets, blockids = rep(1, length(ev_onsets)))
print(evar)
is_continuous(evar)
```

events *Retrieve canonical event information*

Description

Retrieve canonical event information

Usage

```
events(x, drop.empty = FALSE, ...)
```

Arguments

x	The object to summarise.
drop.empty	Logical; whether to drop empty conditions in the resulting factor.
...	Additional arguments passed to methods.

Value

A data.frame (or tibble) with onset, duration, block, and condition columns.

Examples

```
# Create an event term with condition factor
term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)

# Extract canonical event information
evt_info <- events(term)
print(evt_info)
```

Fcontrasts *Compute F-contrasts*

Description

Compute F-contrasts

Usage

```
Fcontrasts(x, ...)

## S3 method for class 'convolved_term'
Fcontrasts(x, ...)

## S3 method for class 'event_model'
Fcontrasts(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Details

Row names of the contrast matrices can specify which levels of the term are tested. Any matching is done against the design matrix column names.

Value

A named list of matrices with F-contrast weights.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)
names(Fcontrasts(emod))
```

---

feature_suffix *Create Feature Suffix*

Description

Generates the f## suffix for multi-column continuous events.

Usage

```
feature_suffix(j, nf)
```

Arguments

j	Integer vector of feature indices (1-based).
nf	Total number of features.

Value

Character vector of suffixes (e.g., `f01`, `f02`).

Examples

```
feature_suffix(1:3, 5)
```

---

get_all_external_hrf_functions *Get All External HRF Function Names*

Description

Returns all function names that should be recognized in formulas from registered external packages.

Usage

```
get_all_external_hrf_functions()
```

Value

Character vector of function names

Examples

```
register_hrfspec_extension(
  spec_class = "demo_hrfspec",
  package = "demoPkg",
  formula_functions = "demo_hrf"
)
get_all_external_hrf_functions()
```

---

get_basis_entry *Look Up a Registered Basis Entry*

Description

Accepts either a single class name or a class vector (the typical output of `class(x)`). Returns the first matching registration entry, walking the class vector from most-specific to least-specific so a subclass's registration takes precedence over its parent.

Usage

```
get_basis_entry(class_name)
```

Arguments

class_name	Character string or character vector of class names.

Value

Registration entry (a list) or NULL if no class is registered.

Examples

```
get_basis_entry("Poly")
```

---

get_external_hrfspec_functions

Get the HRF Function Name for External Specifications

Description

Returns the function name(s) that should be recognized in formulas for a given external HRF specification class.

Usage

```
get_external_hrfspec_functions(spec_class)
```

Arguments

spec_class	Character string naming the class

Value

Character vector of function names, or NULL if not registered

Examples

```
register_hrfspec_extension(
  spec_class = "demo_hrfspec",
  package = "demoPkg",
  formula_functions = c("demo_hrf", "demo_trialwise")
)
get_external_hrfspec_functions("demo_hrfspec")
```

get_external_hrfspec_info *Get Information About a Registered External HRF Specification*

Description

Get Information About a Registered External HRF Specification

Usage

```
get_external_hrfspec_info(spec_class)
```

Arguments

spec_class	Character string naming the class

Value

A list with registration information, or NULL if not registered

Examples

```
register_hrfspec_extension(
  spec_class = "demo_hrfspec",
  package = "demoPkg",
  requires_external_processing = TRUE,
  formula_functions = "demo_hrf"
)
get_external_hrfspec_info("demo_hrfspec")
```

hrf *hemodynamic regressor specification function for model formulas.*

Description

This function is to be used in formulas for fitting functions, e.g. onsets ~ hrf(fac1,fac2) ... It captures the variables/expressions provided and packages them with HRF/contrast information into an hrfspec object, which is then processed by event_model.

Usage

```
hrf(
  ...,
  basis = "spmg1",
  hrf_fun = NULL,
  onsets = NULL,
  durations = NULL,
  prefix = NULL,
  subset = NULL,
  precision = 0.3,
  nbasis = 1,
  contrasts = NULL,
  id = NULL,
  name = NULL,
  lag = 0,
  summate = TRUE,
  normalize = FALSE
)
```

Arguments

`...`	One or more variable names (bare or character) or expressions involving variables present in the `data` argument of `event_model`.
`basis`	the impulse response function or the name of a pre-supplied function, one of: "gamma", "spmg1", "spmg2", "spmg3", "bspline", "gaussian", "tent", "bs". Can also be an `HRF` object.
`hrf_fun`	optional per-onset HRF generator. Can be: • A function that takes a data frame of event data (with columns `onset`, `duration`, `blockid`, plus any term variables) and returns either a single HRF object (recycled to all events) or a list of HRF objects (one per event). • A formula (e.g., `~hrf_column`) referencing a column in the event data that contains a list of pre-built HRF objects. When `hrf_fun` is specified, the `basis` argument is ignored. The generator function is called AFTER any subsetting via `subset=`, ensuring correct alignment between HRFs and events. All returned HRFs must have the same `nbasis` for design matrix consistency.
`onsets`	optional onsets override. If missing, onsets will be taken from the LHS of the main model formula.
`durations`	optional durations override. If missing, durations argument from `event_model` is used.
`prefix`	a character string that is prepended to the variable names and used to identify the term. Can be used to disambiguate two `hrf` terms with the same variable(s) but different onsets or basis functions.
`subset`	an expression indicating the subset of 'onsets' to keep.
`precision`	sampling precision in seconds.

nbasis	number of basis functions – only used for hemodynamic response functions (e.g. bspline) that take a variable number of bases.
contrasts	one or more contrast_spec objects created with the contrast, pair_contrast etc. functions. Must be NULL, a single contrast spec, or a *named* list of contrast specs.
id	a unique character identifier used to refer to term, otherwise will be determined from variable names.
name	Optional human-readable name for the term.
lag	a temporal offset in seconds which is added to onset before convolution
summate	whether impulse amplitudes sum up when duration is greater than 0.
normalize	logical; if TRUE, each convolved regressor column is peak-normalized so that max(abs(column)) == 1. This is useful when events have different durations, since longer events produce taller peaks after convolution. Normalizing ensures that beta estimates reflect condition differences rather than duration-dependent amplitude scaling, which is important for analyses like MVPA or RSA. Default FALSE.

Value

an hrfspec instance

Examples

```
form1 <- onsets ~ hrf(condition, basis="spmg1")

form2 <- onsets ~ hrf(condition) + hrf(RT)

form3 <- onsets ~ hrf(condition, RT, basis="spmg3")

library(rlang)
con1 <- pair_contrast(~ condition == "A", ~ condition == "B", name="AvB")
form4 <- onsets ~ hrf(condition, Poly(RT, 2), contrasts=con1)

form5 <- onsets ~ hrf(condition, hrf_fun = boxcar_hrf_gen())

form6 <- onsets ~ hrf(condition,
                      hrf_fun = weighted_hrf_gen("sub_times", "sub_weights"))
```

Ident *Ident*

Description

A basis that applies identity transform to a set of raw variables.

Usage

```
Ident(...)
```

Arguments

...	a list of variable names

Value

an instance of class `Ident` extending `ParametricBasis`

Examples

```
# Create identity basis from numeric vectors
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 6, 8, 10)
ident_basis <- Ident(x, y)
print(ident_basis$y)
```

---

interaction_contrast *Interaction Contrast*

Description

Create an interaction contrast specification

Usage

```
interaction_contrast(A, name, where = NULL)
```

Arguments

A	A formula specifying the interaction contrast
name	The name of the contrast
where	An optional formula specifying the subset over which the contrast is computed.

Value

An interaction_contrast_spec object containing the specification for generating interaction contrast weights

See Also

`oneway_contrast` for main effects, `pair_contrast` for pairwise comparisons

Examples

```
# Create an interaction contrast for factors A and B
con <- interaction_contrast(~ A * B, name = "A_by_B")

# Create an interaction contrast with a 'where' clause
con <- interaction_contrast(~ A * B, name = "A_by_B",
                            where = ~ block == 1)
```

---

is_categorical *Check if categorical*

Description

Check if categorical

Usage

```
is_categorical(x, ...)

## S3 method for class 'event'
is_categorical(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

Logical scalar indicating whether x is categorical.

Examples

```
# Create a categorical event from factor data
cat_event <- event_factor(
  factor(c("faces", "houses", "faces", "houses")),
  name = "condition",
  onsets = c(0, 10, 20, 30),
  blockids = rep(1, 4)
)
is_categorical(cat_event)  # Returns TRUE

# Create a continuous event from numeric data
cont_event <- event_variable(
  c(1.2, 0.8, 1.5, 0.9),
  name = "reaction_time",
  onsets = c(0, 10, 20, 30),
  blockids = rep(1, 4)
```

```
)
is_categorical(cont_event)  # Returns FALSE

# Event term with mixed types is considered categorical
mixed_term <- event_term(
  list(condition = factor(c("A", "B", "A", "B")),
       modulator = c(1.1, 0.9, 1.2, 0.8)),
  onsets = c(0, 10, 20, 30),
  blockids = rep(1, 4)
)
is_categorical(mixed_term)  # Returns TRUE
```

`is_continuous` *Check if continuous*

Description

Check if continuous

Usage

```
is_continuous(x, ...)

## S3 method for class 'event'
is_continuous(x, ...)
```

Arguments

`x`	The object.
`...`	Additional arguments.

Value

Logical scalar indicating whether `x` is continuous.

Examples

```
# Create a continuous event from numeric vector
cont_event <- event_variable(
  c(1.2, 0.8, 1.5, 0.9),
  name = "reaction_time",
  onsets = c(0, 10, 20, 30),
  blockids = rep(1, 4)
)
is_continuous(cont_event)  # Returns TRUE

# Create a continuous event from matrix
mat_event <- event_matrix(
  matrix(c(1.1, 0.9, 1.2, 0.8, 2.1, 1.9, 2.2, 1.8), nrow = 4),
```

```
  name = "coordinates",
  onsets = c(0, 10, 20, 30),
  blockids = rep(1, 4)
)
is_continuous(mat_event)  # Returns TRUE

# Categorical event is not continuous
cat_event <- event_factor(
  factor(c("faces", "houses", "faces", "houses")),
  name = "condition",
  onsets = c(0, 10, 20, 30),
  blockids = rep(1, 4)
)
is_continuous(cat_event)  # Returns FALSE

# Event term with all continuous events
cont_term <- event_term(
  list(rt = c(1.1, 0.9, 1.2, 0.8),
       accuracy = c(0.95, 0.87, 0.92, 0.88)),
  onsets = c(0, 10, 20, 30),
  blockids = rep(1, 4)
)
is_continuous(cont_term)  # Returns TRUE
```

`is_external_hrfspec` *Check if a Class is a Registered External HRF Specification*

Description

Check if a Class is a Registered External HRF Specification

Usage

```
is_external_hrfspec(x)
```

Arguments

`x`	An object or character string class name

Value

Logical indicating if the class is registered as an external HRF spec

Examples

```
register_hrfspec_extension(
  spec_class = "demo_hrfspec",
  package = "demoPkg"
)
is_external_hrfspec("demo_hrfspec")
```

`labels.event` *Get Formatted Labels for a Single Event*

Description

Returns a character vector of formatted labels for an event object, using the `Variable[Level]` style for categorical events, `Variable[Index]` for multi-column continuous events, or just `Variable` for single continuous events. Useful for getting consistent labels for individual event components. This is distinct from `levels()` which returns the raw level names or column names. Relies on the internal `.level_vector` helper function.

Usage

```
## S3 method for class 'event'
labels(object, ...)
```

Arguments

`object`	An object of class `event`.
`...`	Additional arguments (unused).

Value

A character vector of formatted labels, or `character(0)` if not applicable.

Examples

```
fac <- factor(rep(c("A", "B"), 3))
onsets <- 1:6
ev_fac <- event_factor(fac, "Condition", onsets, blockids = rep(1, length(onsets)))
labels(ev_fac) # Should return c("Condition[A]", "Condition[B]")

vals <- 1:6
ev_num <- event_variable(vals, "Modulator", onsets, blockids = rep(1, length(onsets)))
labels(ev_num) # Should return "Modulator"

mat <- matrix(1:12, 6, 2)
colnames(mat) <- c("C1", "C2")
ev_mat <- event_matrix(mat, "MatrixVar", onsets, blockids = rep(1, length(onsets)))
labels(ev_mat) # Should return c("MatrixVar[1]", "MatrixVar[2]")
```

`levels.Scale` *Extract Levels from fmrireg Objects*

Description

Extract levels from various fmrireg objects. These methods extend the base R `levels` generic to work with fmrireg-specific classes.

Usage

```
## S3 method for class 'Scale'
levels(x, ...)

## S3 method for class 'ScaleWithin'
levels(x, ...)

## S3 method for class 'RobustScale'
levels(x, ...)

## S3 method for class 'event'
levels(x, ...)

## S3 method for class 'event'
columns(x, ...)
```

Arguments

`x`	An object from which to extract levels. Can be: • An `event` object - returns factor levels or column names • A `Scale` object - returns the variable name • A `ScaleWithin` object - returns the variable name • A `RobustScale` object - returns the variable name
`...`	Additional arguments (currently unused).

Value

A character vector of levels or names, depending on the object type:

- For categorical events: the factor levels
- For continuous events: the column names (matrices) or variable name (vectors)
- For scale objects: the variable name being scaled

Functions

- `columns(event)`: Alias for levels.event

Examples

```
# Create a categorical event
fac_event <- event_factor(
  factor(c("A", "B", "A", "B")),
  name = "condition",
  onsets = c(1, 10, 20, 30),
  blockids = rep(1, 4)
)
levels(fac_event)  # Returns: c("A", "B")

# Create a continuous event
cont_event <- event_variable(
  c(1.2, 0.8, 1.5, 0.9),
  name = "reaction_time",
  onsets = c(1, 10, 20, 30),
  blockids = rep(1, 4)
)
levels(cont_event)  # Returns: "reaction_time"
```

---

list_external_hrfspecs

List All Registered External HRF Specifications

Description

List All Registered External HRF Specifications

Usage

```
list_external_hrfspecs()
```

Value

A character vector of registered class names

Examples

```
register_hrfspec_extension(
  spec_class = "demo_hrfspec",
  package = "demoPkg"
)
list_external_hrfspecs()
```

list_registered_bases *List Registered Parametric Basis Classes*

Description

List Registered Parametric Basis Classes

Usage

```
list_registered_bases()
```

Value

Character vector of registered class names.

Examples

```
list_registered_bases()
```

longnames *Extract longnames*

Description

Superseded by `conditions(x, style = "canonical")`.

Usage

```
longnames(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

Character vector of long (fully qualified) names.

Examples

```
# Create a simple event term with one condition factor
term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
longnames(term)  # Returns: "condition.A" "condition.B"

# Create event term with multiple factors
term2 <- event_term(
  list(
    category = factor(c("face", "scene", "face")),
    attention = factor(c("attend", "attend", "ignore"))
  ),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
longnames(term2)
# Returns: "category.face_attention.attend"
#          "category.scene_attention.attend"
#          "category.face_attention.ignore"
```

---

nbasis.hrfspec *Get number of basis functions from hrfspec*

Description

Get number of basis functions from hrfspec

Usage

```
## S3 method for class 'hrfspec'
nbasis(x, ...)
```

Arguments

x	An hrfspec object
...	Additional arguments (unused)

Value

The number of basis functions

Examples

```
# Create hrfspec with canonical HRF (1 basis function)
spec1 <- hrf(condition, basis = "spmg1")
nbasis(spec1)

# Create hrfspec with derivative HRF (2 basis functions)
spec2 <- hrf(condition, basis = "spmg2")
nbasis(spec2)

# Create hrfspec with FIR basis (custom number of basis functions)
spec3 <- hrf(condition, basis = "fir", nbasis = 10)
nbasis(spec3)
```

---

nuisance *Create a Nuisance Specification*

Description

Returns a nuisance term specification from a numeric matrix.

Usage

```
nuisance(x)
```

Arguments

x	A matrix.

Value

An object of class "nuisancespec".

Examples

```
mat <- matrix(rnorm(10), nrow = 5)
nuisance(mat)
```

one_against_all_contrast *One Against All Contrast*

Description

Construct contrasts comparing each factor level against the average of the other levels.

Usage

```
one_against_all_contrast(levels, facname, where = NULL)
```

Arguments

levels	A vector of factor levels to be compared.
facname	A character string specifying the name of the factor containing the supplied levels.
where	An optional formula specifying the subset over which the contrast is computed.

Value

A contrast_set object containing contrasts comparing each factor level against the average of the other levels.

Examples

```
fac <- factor(rep(c("A", "B", "C"), 2))
con <- one_against_all_contrast(levels(fac), "fac")
```

oneway_contrast *One-way Contrast*

Description

Create a one-way contrast specification

Usage

```
oneway_contrast(A, name, where = NULL, basis = NULL, basis_weights = NULL)
```

Arguments

A	A formula specifying the contrast
name	The name of the contrast
where	An optional formula specifying the subset over which the contrast is computed.
basis	NULL (default: use all basis functions), an integer vector specifying which basis function indices to include, or "all". See pair_contrast for details on basis filtering.
basis_weights	NULL (default: equal weights), or a numeric vector of weights to apply to the selected basis functions. Must have the same length as basis selection and will be normalized to sum to 1. See pair_contrast for details on basis weighting.

Value

A oneway_contrast_spec object that can be used to generate contrast weights

See Also

interaction_contrast for testing interactions, pair_contrast for pairwise comparisons

Examples

```
# Create a one-way contrast for a factor 'basis'
con <- oneway_contrast(~ basis, name = "Main_basis")

# Create a one-way contrast with a 'where' clause
con <- oneway_contrast(~ basis, name = "Main_basis",
                       where = ~ block == 1)

# Test only first two basis functions
con <- oneway_contrast(~ condition, name = "Main_early", basis = 1:2)
```

---

pair_contrast *Pair Contrast*

Description

Construct a sum-to-zero contrast between two logical expressions. This function is particularly useful for comparing specific conditions or combinations of conditions.

Usage

```
pair_contrast(A, B, name, where = NULL, basis = NULL, basis_weights = NULL)
```

Arguments

A	A formula representing the first logical expression in the contrast.
B	A formula representing the second logical expression in the contrast.
name	A character string specifying the name of the contrast (mandatory).
where	An optional formula specifying the subset over which the contrast is computed.
basis	NULL (default: use all basis functions), an integer vector specifying which basis function indices to include (e.g., 1, 2:3, c(1,3)), or "all". Only relevant when the HRF uses multiple basis functions (e.g., bspline, FIR, Fourier). Basis indices are 1-based (1 = first basis function, 2 = second, etc.).
basis_weights	NULL (default: equal weights), or a numeric vector of weights to apply to the selected basis functions. Must have the same length as basis selection and will be normalized to sum to 1. Use this to emphasize specific temporal components (e.g., c(.1, .2, .4, .2, .1) for Gaussian-like weighting emphasizing the peak).

Details

The contrast is constructed as (A - B), where A and B are logical expressions that evaluate to TRUE/FALSE for each observation. The resulting contrast weights sum to zero.

When using multi-basis HRFs (e.g., bspline with 5 basis functions), the basis argument allows you to test specific temporal components of the response. For example:

- basis = 1: Test only the first basis function (often the canonical/early response)
- basis = 2:3: Test the second and third basis functions together
- basis = NULL or basis = "all": Test all basis functions (default behavior)

The basis_weights argument allows non-uniform weighting across selected basis functions:

- basis_weights = c(.1, .2, .4, .2, .1): Gaussian-like emphasis on peak
- basis_weights = c(1, 0, 0, 0, 0): Isolate first basis (equivalent to basis = 1)
- Weights are applied within each condition, maintaining contrast sum-to-zero property

Value

A pair_contrast_spec object containing:

A	First logical expression
B	Second logical expression
where	Subsetting formula (if provided)
basis	Basis function specification (if provided)
basis_weights	Basis weights (if provided)
name	Contrast name

See Also

pairwise_contrasts for all pairwise comparisons, contrast_set for creating sets of contrasts

Examples

```
# Compare faces vs scenes (all basis functions)
pair_contrast(~ category == "face", ~ category == "scene", name = "face_vs_scene")

# Test only the second basis function (e.g., linear component for polynomial HRF)
pair_contrast(~ category == "face", ~ category == "scene",
              basis = 2, name = "face_vs_scene_basis2")

# Test early response components (first 3 basis functions)
pair_contrast(~ category == "face", ~ category == "scene",
              basis = 1:3, name = "face_vs_scene_early")

# Compare with subsetting
pair_contrast(~ category == "face", ~ category == "scene",
              name = "face_vs_scene_block1",
              where = ~ block == 1)

# Complex logical expressions
pair_contrast(~ stimulus == "face" & emotion == "happy",
              ~ stimulus == "face" & emotion == "sad",
              name = "happy_vs_sad_faces")
```

---

pairwise_contrasts *Pairwise Contrasts*

Description

Construct pairwise contrasts for all combinations of factor levels.

Usage

```
pairwise_contrasts(levels, facname, where = NULL, name_prefix = "con")
```

Arguments

levels	A vector of factor levels to be compared.
facname	The name of the factor variable (column name in the design) these levels belong to.
where	An optional formula specifying the subset over which the contrast is computed.
name_prefix	A character string to prefix the generated contrast names (default: "con").

Value

A contrast_set object containing pairwise contrasts for all combinations of factor levels.

Examples

```
# Assuming 'my_factor' is a column name
pairwise_contrasts(c("A", "B", "C"), facname = "my_factor")
pairwise_contrasts(c("A", "B", "C"), facname = "my_factor", name_prefix = "pair")
```

`plot.event_model` *Plot Event Model*

Description

Creates a line plot visualization of the predicted BOLD response for each regressor in an event_model object.

Usage

```
## S3 method for class 'event_model'
plot(
  x,
  term_name = NULL,
  facet_threshold = Inf,
  label_mode = c("auto", "compact", "none"),
  max_labels = 30,
  abbrev_min = 10,
  strip_text_size = 8,
  block_x = c("global", "run"),
  facet_by_block = FALSE,
  show_block_bounds = TRUE,
  ...
)
```

Arguments

`x`	An `event_model` object.
`term_name`	Character. Name of specific term to plot. If NULL, plots all terms.
`facet_threshold`	Integer. Switch to faceting when number of regressors exceeds this value. Default 6.
`label_mode`	Character. One of `"auto"`, `"compact"`, `"none"`. In `"auto"` mode the method abbreviates labels for moderate counts and suppresses labels entirely when they are excessive (> `max_labels`). `"compact"` always abbreviates labels. `"none"` suppresses legend and facet strip labels.
`max_labels`	Integer. When `label_mode = "auto"` and the number of regressors exceeds this value, labels are suppressed. Default 30.
`abbrev_min`	Integer. Minimum length used by `base::abbreviate()` when compacting labels. Default 10.

`strip_text_size`
: Numeric. Strip label text size when faceting with labels. Default 8.

`block_x`
: Time axis to use for multi-run designs. `"global"` (default) uses concatenated time so each block occupies a distinct x-range; `"run"` uses run-relative time that restarts each block. In either case line segments are grouped by block so a regressor is never connected across a run boundary (this is what prevents the spurious high-frequency oscillations that appear when all blocks share one block-relative axis).

`facet_by_block`
: Logical; if `TRUE`, draw one panel per block. Defaults to `FALSE`. Useful for multi-run designs where overlaid runs are cluttered.

`show_block_bounds`
: Logical; if `TRUE` (default), draw dashed vertical rules at each run's start/end (`blocklen * TR`). These make late starts and overruns obvious and complement the `event_model()` onset bounds check. Drawn only when a `sampling_frame` is available.

`...`
: Additional arguments (currently unused).

Details

This method attempts to keep labels readable when there are many regressors (e.g., trial-wise designs) by switching to faceting and either abbreviating or suppressing labels depending on thresholds. You can control this behavior via `label_mode`, `max_labels`, and `abbrev_min`.

Value

A ggplot2 object showing the predicted BOLD timecourses.

Examples

```
# Create a simple event model
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)

# Plot all regressors
plot(emod)

# Plot specific term only
plot(emod, term_name = "cond")
```

plot_contrasts *plot_contrasts*

Description

Generic function for plotting contrasts.

Usage

```
plot_contrasts(x, ...)
```

Arguments

x	Object containing contrast information
...	Additional arguments passed to methods

Value

A plot object (typically ggplot2) displaying the contrasts. The exact type depends on the method used.

Examples

```
# Create example data
des <- data.frame(
  onset = c(1, 3, 5, 7),
  cond = factor(c("A", "B", "A", "B")),
  run = c(1, 1, 1, 1)
)

# Create sampling frame and event model
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)

# Create contrast set
cset <- contrast_set(
  main_A = unit_contrast(~ cond == "A", name = "A_vs_baseline"),
  diff = pair_contrast(~ cond == "A", ~ cond == "B", name = "A_vs_B")
)

# Create event model with contrasts
emod <- event_model(onset ~ hrf(cond, contrasts = cset),
                    data = des, block = ~run, sampling_frame = sframe)

# Plot the contrasts
plot_contrasts(emod)
```

plot_contrasts.event_model *plot_contrasts.event_model*

Description

Produces a heatmap of all contrasts defined for an `event_model`. Rows = each contrast (or column of an F-contrast), columns = each regressor in the full design matrix, and the fill color = the contrast weight.

Usage

```
## S3 method for class 'event_model'
plot_contrasts(
  x,
  absolute_limits = FALSE,
  rotate_x_text = TRUE,
  scale_mode = c("auto", "diverging", "one_sided"),
  coord_fixed = TRUE,
  ...
)
```

Arguments

`x`	An `event_model` with (lazily) defined contrasts.
`absolute_limits`	Logical; if `TRUE`, the color scale is fixed at (-1,1). If `FALSE`, the range is set to (min, max) of the weights.
`rotate_x_text`	Logical; if `TRUE`, rotate x-axis labels for readability.
`scale_mode`	Character; 'auto', 'diverging', or 'one_sided' color scaling.
`coord_fixed`	Logical; if TRUE, use fixed aspect ratio.
`...`	Further arguments passed to `geom_tile`, e.g. `color="grey80"`.

Value

A `ggplot2` object (a heatmap).

Examples

```
# Create event model with contrasts
des <- data.frame(
  onset = c(0, 10, 20, 30, 40, 50),
  run = 1,
  cond = factor(c("A", "B", "C", "A", "B", "C"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 60, TR = 1)
```

```
cset <- contrast_set(
  A_vs_B = pair_contrast(~ cond == "A", ~ cond == "B", name = "A_vs_B"),
  B_vs_C = pair_contrast(~ cond == "B", ~ cond == "C", name = "B_vs_C")
)
emod <- event_model(onset ~ hrf(cond, contrasts = cset),
                    data = des, block = ~run, sampling_frame = sframe)
plot_contrasts(emod)
```

`Poly` *Polynomial basis*

Description

Orthogonal polynomial expansion of a linear term based on `poly`

Usage

```
Poly(x, degree)
```

Arguments

`x`	a numeric vector at which to evaluate the polynomial. Missing values are not allowed in x.
`degree`	the degree of the polynomial. Must be less than the number of unique points.

Value

an instance of class `Poly` extending `ParametricBasis`

See Also

poly

Examples

```
# Create a 3rd degree polynomial basis
x_vals <- c(1, 2, 3, 4, 5, 6)
poly_basis <- Poly(x_vals, degree = 3)
print(poly_basis$y)
```

poly_contrast *Polynomial Contrast*

Description

Create polynomial contrasts for testing trends across ordered factor levels. This is particularly useful for analyzing factors with a natural ordering (e.g., time, dose).

Usage

```
poly_contrast(
  A,
  name,
  where = NULL,
  degree = 1,
  value_map = NULL,
  basis = NULL,
  basis_weights = NULL
)
```

Arguments

A	A formula specifying the ordered factor.
name	A character string identifying the contrast.
where	An optional formula for subsetting the data.
degree	An integer specifying the degree of the polynomial (default: 1).
value_map	An optional list mapping factor levels to numeric values.
basis	NULL (default: use all basis functions), an integer vector specifying which basis function indices to include, or "all". See pair_contrast for details on basis filtering.
basis_weights	NULL (default: equal weights), or a numeric vector of weights to apply to the selected basis functions. Must have the same length as basis selection and will be normalized to sum to 1. See pair_contrast for details on basis weighting.

Details

The function creates orthogonal polynomial contrasts up to the specified degree. These contrasts can test for linear, quadratic, cubic, and higher-order trends in the data. The value_map parameter allows for non-uniform spacing between levels.

Value

A poly_contrast_spec object containing the specification for generating polynomial contrast weights.

See Also

oneway_contrast for categorical contrasts, interaction_contrast for interaction effects

Examples

```
# Linear trend across time points
pcon <- poly_contrast(~ time, name = "linear_time", degree = 1)

# Cubic trend with custom spacing
pcon <- poly_contrast(~ dose, name = "dose_cubic",
                      degree = 3,
                      value_map = list("low" = 0, "med" = 2, "high" = 5))

# Linear trend for only first basis function
pcon <- poly_contrast(~ dose, name = "dose_linear_basis1",
                      degree = 1, basis = 1)
```

---

predict.ParametricBasis

Predict from a ParametricBasis object

Description

Dispatch to the appropriate method for transforming new data according to a specific parametric basis.

Usage

```
## S3 method for class 'ParametricBasis'
predict(object, newdata, ...)

## S3 method for class 'Standardized'
predict(object, newdata, ...)

## S3 method for class 'Poly'
predict(object, newdata, ...)

## S3 method for class 'BSpline'
predict(object, newdata, ...)

## S3 method for class 'Ident'
predict(object, newdata, ...)

## S3 method for class 'Scale'
predict(object, newdata, ...)

## S3 method for class 'ScaleWithin'
```

```
predict(object, newdata, newgroup, ...)

## S3 method for class 'RobustScale'
predict(object, newdata, ...)
```

Arguments

object	ParametricBasis object.
newdata	Numeric vector to transform.
...	Additional arguments.
newgroup	Optional factor for group-dependent bases.

Value

A numeric matrix with transformed values (one column per basis component).

Examples

```
# Create polynomial basis from training data
train_x <- 1:10
poly_basis <- Poly(train_x, degree = 2)

# Predict on new data
new_x <- c(5.5, 7.3, 11.2)
predict(poly_basis, new_x)

# Create scaling basis
train_vals <- c(10, 20, 30, 40, 50)
scale_basis <- Scale(train_vals)

# Apply same scaling to new data
new_vals <- c(15, 25, 35)
predict(scale_basis, new_vals)
```

---

print.baseline_model *Print a Baseline Model*

Description

Displays key information about the baseline model components and a preview of the design matrix.

Print a contrast set.

Print a contrast specification.

Print a contrast.

Print a polynomial contrast specification.

Print a contrast difference specification.

Provides a concise summary of an event object using cli.

Provides a concise summary of an `event_model` object using `cli`.

Provides a concise summary of an event_term object using cli.

Usage

```
## S3 method for class 'baseline_model'
print(x, ...)

## S3 method for class 'contrast_set'
print(x, ...)

## S3 method for class 'contrast_spec'
print(x, ...)

## S3 method for class 'contrast'
print(x, ...)

## S3 method for class 'poly_contrast_spec'
print(x, ...)

## S3 method for class 'contrast_diff_spec'
print(x, ...)

## S3 method for class 'event'
print(x, ...)

## S3 method for class 'event_model'
print(x, ...)

## S3 method for class 'fmri_term'
print(x, ...)

## S3 method for class 'convolved_term'
print(x, ...)

## S3 method for class 'event_term'
print(x, ...)
```

Arguments

x	An event_term object.
...	Additional arguments (unused).

Value

The input object, invisibly.

Examples

```
sframe <- fmrihrf::sampling_frame(blocklens = 5, TR = 1)
bmod <- baseline_model(sframe = sframe)
print(bmod)
```

print.sampling_frame *Print and Plot for sampling_frame*

Description

Custom print and plot methods for sampling_frame objects (from fmrihrf). The print method provides a concise summary focused on runs/blocks, TR and total scans without mentioning lower-level evaluation precision. The plot method visualizes runs over time.

Usage

```
## S3 method for class 'sampling_frame'
print(x, ...)

## S3 method for class 'sampling_frame'
plot(x, style = c("timeline", "grid"), show_ticks = FALSE, tick_every = 5, ...)
```

Arguments

x	A sampling_frame object created by fmrihrf::sampling_frame().
...	Unused.
style	Plot style for plot(). One of "timeline" (default) or "grid". "timeline" draws a horizontal bar per run; "grid" shows a scan grid by run.
show_ticks	Logical; for plot(), whether to show per-TR tick marks along each run (timeline style only). Default FALSE.
tick_every	Integer; draw a tick every tick_every TRs when show_ticks = TRUE. Default 5.

Value

For print(), returns x invisibly. For plot(), a ggplot object.

Examples

```
sf <- fmrihrf::sampling_frame(blocklens = c(60, 120), TR = 2)
print(sf)
plot(sf)
```

`register_basis` *Register a Parametric Basis Class*

Description

Register a basis class so that the term-naming pipeline produces a well-formed term tag (`<prefix>_<varname>`) and so that design-matrix metadata identifies columns built from this basis as `parametric`. Built-in bases (`Poly`, `BSpline`, `Scale`, `Standardized`, `ScaleWithin`, `RobustScale`, `Ident`) are registered automatically.

Usage

```
register_basis(
  class_name,
  prefix = NULL,
  modulation = c("parametric", "amplitude"),
  description = NULL
)
```

Arguments

`class_name`	Character string naming the S3 class extending `ParametricBasis`.
`prefix`	Character string used as the term-tag prefix for terms whose sole variable is `class_name(var, ...)`. Use `NULL` (or omit) if the basis should not contribute a tag prefix (e.g., `Ident`).
`modulation`	Character string describing the modulation kind for columns generated by this basis. Defaults to `"parametric"`. Use `"amplitude"` for an unmodulated identity-style basis.
`description`	Optional human-readable description.

Value

Invisibly returns the registration entry.

Examples

```
# Register a custom orthogonal-polynomial basis with prefix "ortho"
register_basis("OrthoPoly", prefix = "ortho")
```

`register_hrfspec_extension` *Register an External HRF Specification Type*

Description

Register a new HRF specification class that can be used in event models. This allows external packages to extend fmridesign with their own HRF types.

Usage

```
register_hrfspec_extension(
  spec_class,
  package,
  convolved_class = NULL,
  requires_external_processing = FALSE,
  formula_functions = NULL
)
```

Arguments

`spec_class`	Character string naming the class to register
`package`	Character string naming the package providing the class
`convolved_class`	Optional character string naming the associated convolved term class
`requires_external_processing`	Logical indicating if this spec should be skipped during standard convolution (e.g., for AFNI terms that are processed externally)
`formula_functions`	Optional character vector of function names that should be recognised in formulas and mapped to this HRF specification class.

Value

Invisible NULL

Examples

```
register_hrfspec_extension(
  spec_class = "afni_hrfspec",
  package = "afnireg",
  convolved_class = "afni_hrf_convolved_term",
  requires_external_processing = TRUE,
  formula_functions = "afni_hrf"
)
```

`regressors` *Extract regressors*

Description

Convolve the event-term design matrix with an HRF and return the resulting regressors.

Usage

```
regressors(x, ...)

## S3 method for class 'event_term'
regressors(x, hrf, sampling_frame, summate = FALSE, drop.empty = TRUE, ...)
```

Arguments

`x`	The object.
`...`	Additional arguments.
`hrf`	HRF function
`sampling_frame`	sampling_frame object
`summate`	Logical; sum HRF responses
`drop.empty`	Logical; drop empty conditions

Value

Character vector of regressor names for `x`.

Examples

```
# Create an event term with two conditions
term <- event_term(
  list(condition = factor(c("A", "B", "A", "B"))),
  onsets = c(0, 10, 20, 30),
  blockids = c(1, 1, 1, 1)
)

# Create a sampling frame for timing information
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 2)

# Extract regressors convolved with canonical HRF
reg <- regressors(term, hrf = fmrihrf::HRF_SPMG1, sampling_frame = sframe)
names(reg)  # Shows regressor names: "condition.A" "condition.B"
```

requires_external_processing *Check if an Object Requires External Processing*

Description

Determines if an HRF specification or convolved term should be handled by external tools rather than R's standard convolution.

Usage

```
requires_external_processing(x)
```

Arguments

x	An object to check

Value

Logical indicating if external processing is required

Examples

```
register_hrfspec_extension(
  spec_class = "demo_hrfspec",
  package = "demoPkg",
  requires_external_processing = TRUE
)
requires_external_processing("demo_hrfspec")
```

residualize *Residualize Data Against a Design*

Description

Projects the data onto the orthogonal complement of the design, returning residuals from an OLS fit. Specialized for `event_model`, `baseline_model`, or a raw numeric design matrix.

Usage

```
residualize(x, data, cols = NULL, ...)
```

Arguments

x	A design object (event_model, baseline_model) or a numeric matrix (design matrix X).
data	A numeric vector/matrix/data.frame of observations Y with rows matching nrow(design_matrix(x)).
cols	Optional integer or character vector selecting columns of the design to project out.
...	Additional arguments passed to methods.

Value

Residuals with the same dimensions as data.

Examples

```
# Simple example with a raw design matrix
X <- cbind(1, 1:5)
Y <- cbind(1:5, 2:6)
R <- residualize(X, Y)
dim(R)
```

---

residualize-methods *Residualize methods*

Description

S3 methods for the residualize() generic. These methods project data onto the orthogonal complement of a design, returning OLS residuals.

Usage

```
## S3 method for class 'matrix'
residualize(x, data, cols = NULL, ...)

## S3 method for class 'event_model'
residualize(x, data, cols = NULL, ...)

## S3 method for class 'baseline_model'
residualize(x, data, cols = NULL, ...)
```

Arguments

x	A design object: matrix, event_model, or baseline_model.
data	Numeric vector/matrix/data.frame of observations Y.
cols	Optional integer or character vector selecting columns to project out.
...	Additional arguments (currently unused).

Value

A numeric matrix of residuals with the same dimensions as `data`.

Examples

```
# Residualize with a raw matrix
X <- cbind(1, 1:10)
Y <- matrix(rnorm(20), ncol = 2)
R <- residualize(X, Y)
dim(R)  # 10 x 2

# Residualize with an event model
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run,
                    sampling_frame = sframe)
Y_sim <- matrix(rnorm(40 * 2), ncol = 2)
R_emod <- residualize(emod, Y_sim)
dim(R_emod)  # 40 x 2
```

RobustScale *Robust Scaling (Median/MAD)*

Description

Robust Scaling (Median/MAD)

Usage

```
RobustScale(x)
```

Arguments

`x`	numeric vector (NAs allowed)

Value

object of class c("RobustScale","ParametricBasis")

Examples

```
# Create a robust scale transformed basis using median and MAD
x_vals <- c(1, 2, 3, 4, 100)  # Note the outlier
robust_basis <- RobustScale(x_vals)
print(robust_basis$y)
print(robust_basis$median)
print(robust_basis$mad)
```

`sanitize` *Sanitize Strings for Use in R Names*

Description

Wraps `make.names` but allows control over dot replacement.

Usage

```
sanitize(x, allow_dot = TRUE)
```

Arguments

`x`	A character vector.
`allow_dot`	Logical, if `FALSE`, dots (`.`) are replaced with underscores (_).

Value

A sanitized character vector.

Examples

```
sanitize("a.b c")
sanitize("a.b c", allow_dot = FALSE)
```

`Scale` *Z-score (global) basis*

Description

Z-score (global) basis

Usage

```
Scale(x)
```

Arguments

`x`	numeric vector (NAs allowed)

Value

object of class c("Scale","ParametricBasis")

Examples

```
# Create a z-score transformed basis
x_vals <- c(1, 3, 5, 7, 9, 11)
scale_basis <- Scale(x_vals)
print(scale_basis$y)
print(scale_basis$mean)
print(scale_basis$sd)
```

ScaleWithin *Z-score within groups*

Description

Z-score within groups

Usage

```
ScaleWithin(x, g)
```

Arguments

x	numeric vector
g	grouping factor / character / integer of same length as x

Value

An object of class `ScaleWithin` (a `ParametricBasis`).

Examples

```
# Create a within-group z-score transformed basis
x_vals <- c(1, 2, 3, 10, 11, 12)
groups <- c("A", "A", "A", "B", "B", "B")
scale_within_basis <- ScaleWithin(x_vals, groups)
print(scale_within_basis$y)
print(scale_within_basis$means)
print(scale_within_basis$sds)
```

shortnames *Extract shortnames*

Description

Superseded by `conditions(x, style = "display")`.

Usage

```
shortnames(x, ...)
```

Arguments

`x`	The object.
`...`	Additional arguments.

Value

Character vector of short names.

Examples

```
# Create a simple event term with one condition factor
term <- event_term(
  list(condition = factor(c("A", "B", "A"))),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
shortnames(term)  # Returns: "A" "B"

# Create event term with multiple factors
term2 <- event_term(
  list(
    category = factor(c("face", "scene", "face")),
    attention = factor(c("attend", "attend", "ignore"))
  ),
  onsets = c(0, 10, 20),
  blockids = c(1, 1, 1)
)
shortnames(term2)  # Returns: "face:attend" "scene:attend" "face:ignore"
```

`sliding_window_contrasts` *Sliding-Window Contrasts (Disjoint)*

Description

Generate a set of A-vs-B contrasts where A and B are adjacent, equally sized and disjoint windows over an ordered factor. For window size k, contrast i compares A = `levels[i:(i+k-1)]` against B = `levels[(i+k):(i+2k-1)]`. This yields `length(levels) - 2*k + 1` contrasts that detect local changes across the sequence without overlapping masks.

Usage

```
sliding_window_contrasts(
  levels,
  facname,
  window_size = 2,
  where = NULL,
  name_prefix = "win"
)
```

Arguments

`levels`	Character vector of ordered factor levels.
`facname`	Name of the factor (column in the design).
`window_size`	Positive integer window size (default 2).
`where`	Optional formula to subset events used when computing weights.
`name_prefix`	Prefix for generated contrast names (default "win").

Value

A `contrast_set` of `pair_contrast` specifications.

Examples

```
# For levels 1..5, generate 2 disjoint adjacent-window contrasts (k=2)
sliding_window_contrasts(as.character(1:5), facname = "intensity", window_size = 2)

# For k=3 with 7 levels (disjoint windows):
# A=[1,2,3] vs B=[4,5,6], then A=[2,3,4] vs B=[5,6,7]
sliding_window_contrasts(LETTERS[1:7], facname = "difficulty", window_size = 3)
```

split_by_block *Split by block*

Description

Split by block

Usage

```
split_by_block(x, ...)

## S3 method for class 'event_model'
split_by_block(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

A list split by block/run.

Examples

```
des <- data.frame(
  onset = c(0, 10, 20, 30, 5, 15, 25, 35),
  run = c(1, 1, 1, 1, 2, 2, 2, 2),
  cond = factor(c("A", "B", "A", "B", "A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = c(40, 40), TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)
block_list <- split_by_block(emod)
length(block_list)
```

split_onsets *Split onsets*

Description

Split onsets

Usage

```
split_onsets(x, sframe, global = FALSE, blocksplit = FALSE, ...)

## S3 method for class 'event_term'
split_onsets(x, sframe, global = FALSE, blocksplit = FALSE, ...)
```

Arguments

x	The object.
sframe	The sampling frame object containing timing information.
global	Whether onsets are in global time units (across all runs).
blocksplit	Whether to split onsets by blocks.
...	Additional arguments.

Value

A list of onset vectors, one per block (unless global=TRUE).

Examples

```
# Create an event term with mixed conditions across blocks
conditions <- factor(c("A", "B", "A", "B", "A", "B"))
onsets <- c(5, 15, 25, 105, 115, 125)  # Events in blocks 1 and 2
blockids <- c(1, 1, 1, 2, 2, 2)

term <- event_term(
  list(condition = conditions),
  onsets = onsets,
  blockids = blockids
)

# Create sampling frame for two blocks of 50 TRs each
sframe <- fmrihrf::sampling_frame(blocklens = c(50, 50), TR = 2)

# Split onsets by condition (default behavior)
onset_list <- split_onsets(term, sframe)
names(onset_list)  # Shows condition names
onset_list$condition.A  # Onsets for condition A

# Split with global timing (onsets relative to start of experiment)
global_onsets <- split_onsets(term, sframe, global = TRUE)

# Split by both condition and block
block_split <- split_onsets(term, sframe, blocksplit = TRUE)
```

Standardized *Standardized basis*

Description

Standardize a numeric vector by centering and scaling, handling NAs appropriately. If the computed standard deviation is NA or zero, a small constant (1e-6) is used instead to avoid division by zero. The returned basis matrix has one column with this standardized name.

Usage

```
Standardized(x)
```

Arguments

x	a numeric vector to standardize. Missing values are allowed and will be replaced with 0 after standardization.

Value

an instance of class `Standardized` extending `ParametricBasis`

Examples

```
# Standardize a numeric vector
x_vals <- c(10, 20, 30, 40, 50)
std_basis <- Standardized(x_vals)
print(std_basis$y)
print(std_basis$mean)
print(std_basis$sd)
```

---

sub_basis *sub_basis*

Description

Subset a parametric basis regressor.

Usage

```
sub_basis(x, subset)

## S3 method for class 'Scale'
sub_basis(x, subset)

## S3 method for class 'ScaleWithin'
sub_basis(x, subset)

## S3 method for class 'RobustScale'
sub_basis(x, subset)
```

Arguments

x	the object
subset	the subset (logical or integer indices)

Value

An object of the same class as x with subset applied.

Examples

```
# Create some sample data
x_vals <- 1:10
rt_vals <- rnorm(10, 500, 50)

# Create different basis objects
poly_basis <- Poly(x_vals, degree = 3)
scale_basis <- Scale(rt_vals)
bspline_basis <- BSpline(x_vals, degree = 2)

# Subset with integer indices
poly_sub <- sub_basis(poly_basis, 1:5)
scale_sub <- sub_basis(scale_basis, c(1, 3, 5, 7, 9))

# Subset with logical indices
logical_idx <- x_vals <= 5
bspline_sub <- sub_basis(bspline_basis, logical_idx)

# Check dimensions
nrow(poly_basis$y)  # 10
nrow(poly_sub$y)    # 5
nrow(scale_sub$y)   # 5
nrow(bspline_sub$y) # 5
```

---

term_indices *Extract term indices*

Description

Extract term indices

Usage

```
term_indices(x, ...)

## Default S3 method:
term_indices(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

Integer vector or list mapping term(s) to column indices.

Examples

```
# Create a sampling frame and event model
sf <- fmrihrf::sampling_frame(blocklens = c(100, 100), TR = 2)
events <- data.frame(
  onset = c(10, 30, 50, 70),
  condition = c("A", "B", "A", "B"),
  block = c(1, 1, 2, 2)
)
model <- event_model(onset ~ hrf(condition), events, ~ block, sf)

# Get design matrix and extract term indices
dm <- design_matrix(model)
indices <- term_indices(dm)
print(indices)

# Access indices for specific term
condition_indices <- indices[["condition"]]
print(condition_indices)
```

---

term_matrices *Extract term matrices*

Description

Extract term matrices

Usage

```
term_matrices(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

A list of matrices/tibbles, one per term.

Examples

```
# Create a simple experimental design with event model
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)

# Extract term matrices - returns list with one matrix per term
term_mats <- term_matrices(emod)
names(term_mats)     # Shows term names
ncol(term_mats[[1]]) # Number of columns for first term

# Create baseline model and extract its term matrices
bmod <- baseline_model(sframe = sframe)
baseline_mats <- term_matrices(bmod)
names(baseline_mats) # Shows baseline term names
```

---

term_names *Extract term names*

Description

Extract term names

Usage

```
term_names(x, ...)
```

Arguments

x	The object.
...	Additional arguments.

Value

Character vector of term names.

Examples

```
# Create sample event data
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
```

```
# Event model example
emod <- event_model(onset ~ hrf(cond), data = des, block = ~run, sampling_frame = sframe)
term_names(emod)  # Returns "cond"

# Baseline model example
bmod <- baseline_model(basis = "poly", degree = 3, sframe = sframe)
term_names(bmod)  # Returns c("constant", "baseline_poly_3")
```

`trialwise` *trialwise*

Description

Generate one regressor per trial (plus an optional grand-mean column) by delegating everything to `hrf()`.

Usage

```
trialwise(
  basis = "spmg1",
  lag = 0,
  nbasis = 1,
  add_sum = FALSE,
  label = "trial",
  durations = NULL,
  normalize = FALSE
)
```

Arguments

`basis`, `lag`, `nbasis`	Passed straight to `hrf()`.
`add_sum`	If TRUE, append a column that is the average of all trialwise columns (useful as a conventional main effect).
`label`	Term label / prefix for the generated columns.
`durations`	Optional durations override (passed to `hrf()`). If NULL, uses the durations argument from `event_model`.
`normalize`	logical; if TRUE, each trialwise regressor column is peak-normalized so that `max(abs(column)) == 1`. When events have different durations, longer events produce taller peaks after convolution. Normalizing equalizes amplitudes so that beta estimates are comparable across trials regardless of duration. Particularly useful for MVPA and RSA analyses. Default FALSE.

Details

Use it **only on the RHS** of an event-model formula:

```
onset ~ trialwise(basis = "spmg1", add_sum = TRUE)
```

Value

An hrfspec term to be used on the RHS of an event-model formula.

Examples

```
# Create example trial data for beta-series analysis
trial_data <- data.frame(
  onset = c(2, 8, 14, 20, 26),
  run = c(1, 1, 1, 1, 1)
)

# Create sampling frame (30 TRs, TR=2s)
sframe <- fmrihrf::sampling_frame(blocklens = 30, TR = 2)

# Basic trialwise model - creates one regressor per trial
emod_trials <- event_model(onset ~ trialwise(),
                           data = trial_data,
                           block = ~run,
                           sampling_frame = sframe)
print(emod_trials)

# Trialwise with different basis and grand mean
emod_trials_mean <- event_model(onset ~ trialwise(basis = "spmg2", add_sum = TRUE),
                                data = trial_data,
                                block = ~run,
                                sampling_frame = sframe)
print(emod_trials_mean)
```

---

unit_contrast *Unit Contrast*

Description

Construct a contrast that sums to 1 and is used to define contrasts against the baseline.

Usage

```
unit_contrast(A, name, where = NULL)
```

Arguments

A	A formula representing the contrast expression.
name	A character string specifying the name of the contrast.
where	An optional formula specifying the subset of conditions to apply the contrast to.

Value

A unit_contrast_spec object containing the contrast that sums to 1.

Examples

```
# Test main effect of Face against baseline
con <- unit_contrast(~ Face, name="Main_face")

# Test main effect within specific blocks
con2 <- unit_contrast(~ Face, name="Face_early", where = ~ block <= 3)
```

`validate_contrasts` *Validate contrast weights against a design matrix or event model*

Description

Provides basic diagnostics for t- and F-contrasts once the design matrix is available. You can either pass an `event_model` (to validate all attached contrasts) or a design matrix plus custom weights.

Usage

```
validate_contrasts(x, weights = NULL, tol = 1e-08)
```

Arguments

`x`	An `event_model` or a numeric matrix/data.frame design matrix.
`weights`	Optional contrast weights. May be a numeric vector (t-contrast), a numeric matrix (F-contrast with columns as contrast vectors), or a named list mapping names to vectors/matrices. If `NULL` and `x` is an `event_model`, all attached t- and F-contrasts are validated.
`tol`	Numeric tolerance for zero checks. Default `1e-8`.

Details

Checks include:

- Estimability: whether each contrast column lies in the row space of `X`.
- Sum-to-zero: whether the weights sum to ~0 (t-contrasts only).
- Intercept orthogonality: whether weights on intercept-like columns are ~0.
- Full-rank (F only): whether an F-contrast matrix has full column rank.

Value

A data.frame with one row per validated contrast column and the following columns: `name`, `type` ("t" or "F"), `estimable`, `sum_to_zero`, `orthogonal_to_intercept`, `full_rank` (F only), and `nonzero_weights`.

Examples

```
# Create a simple event model
des <- data.frame(
  onset = c(0, 10, 20, 30),
  run = 1,
  cond = factor(c("A", "B", "A", "B"))
)
sframe <- fmrihrf::sampling_frame(blocklens = 40, TR = 1)
emodel <- event_model(onset ~ hrf(cond), data = des, block = ~run,
                      sampling_frame = sframe)

# Validate all attached contrasts on a model
res <- validate_contrasts(emodel)

# Validate a custom vector against a model
v <- rep(0, ncol(design_matrix(emodel)))
v[1] <- 1
v[2] <- -1
res2 <- validate_contrasts(emodel, weights = v)
```

---

weighted_hrf_gen *Create weighted HRF generator from list columns*

Description

Creates a generator function for use with the `hrf_fun` parameter in `hrf()`. The generator produces weighted impulse HRFs from columns containing lists of sub-event times and weights.

Usage

```
weighted_hrf_gen(
  times_col = "sub_times",
  weights_col = "sub_weights",
  relative = FALSE,
  method = "constant",
  normalize = FALSE
)
```

Arguments

`times_col`	Character; name of the column containing sub-event times (relative or absolute).
`weights_col`	Character; name of the column containing sub-event weights.
`relative`	Logical; if TRUE, times are relative to event onset; if FALSE, times are absolute and will be converted to relative by subtracting the onset. Default FALSE (absolute times).
`method`	Character; interpolation method for `hrf_weighted()`. Default "constant".
`normalize`	Logical; whether to normalize the weighted HRF. Default FALSE.

Value

A function that takes an event data frame and returns a list of HRF objects.

See Also

boxcar_hrf_gen() for duration-based boxcar HRFs

Examples

```
trial_data <- data.frame(
  onset = c(0, 20),
  sub_times = I(list(c(0, 1, 2), c(0, 3, 6))),
  sub_weights = I(list(c(0.2, 0.5, 0.3), c(0.1, 0.6, 0.3))),
  run = 1
)
sf <- fmrihrf::sampling_frame(blocklens = 50, TR = 2)

emod <- event_model(
 onset ~ hrf(onset, hrf_fun = weighted_hrf_gen("sub_times", "sub_weights", relative = TRUE)),
  data = trial_data, block = ~run, sampling_frame = sf
)
print(emod)
```

Index